FINAL EVALUATION REPORT
Microsoft Corporation
SQL Server 2000
Version 8.0

Science Applications International Corporation
Center for Information Security Technology Lab
7125 Columbia Gateway Drive, Suite 300
Columbia, MD 21046

August 11, 2000

Table of Contents

List Of Figures

List of Tables

1 Introduction

In November 1998, Science Applications International Corporation (SAIC) began a product evaluation of Microsoft SQL Server 2000 Version 8.0 running on Windows NT Server and Workstation Versions 4.0 Service Pack 6a and C2 Update, all products of the Microsoft Corporation. This report gives evidence and analysis of the security features and assurances provided by Microsoft SQL Server 2000 Version 8.0. This report documents the evaluation team's understanding of the product's security design and appraises its functions and integrity against the C Division security requirements of the Trusted Database Interpretation (TDI) of the Trusted Computer System Evaluation Criteria (TCSEC). Material for this report was gathered by the SAIC evaluation team through documentation, interaction with system developers, and testing.

1.1 Evaluation Process Overview

The Trust Technology Assessment Program (TTAP) evaluation process consists of two high-level phases: Pre-Evaluation and Evaluation. Pre-Evaluation consists of those activities that are recommended for a TTAP evaluation facility (TEF) to complete prior to beginning an evaluation to ensure that the product and its associated evaluation evidence are ready for evaluation. An inadequately tested product or incomplete documentation can substantially delay the schedule and increase the cost of an evaluation. Because the pre-evaluation activities occur only between the TEF and the vendor, TTAP imposes no requirements during this phase.

Evaluation begins with a formal agreement between the vendor and the TEF. All contract negotiations, including any discussions regarding the price of an evaluation, the nature or conditions of payment, and the schedule for the evaluation, are left entirely up to the TEF and the vendor. After the vendor and the TEF sign a contract, the TEF submits the name of the vendor, the name and type of product, the schedule, and the members of the evaluation team to the TTAP Oversight Board. The evaluation may begin as soon as the contract is signed. Note that no authorization to proceed is needed from the Oversight Board; the notification is simply to allow the board to allocate and schedule Technical Review Board (TRB) resources.

Work on the evaluation begins after the agreement is signed. The evaluation team must determine that the system meets all of the TCSEC C2 requirements, and prepare and defend an Initial Product Assessment Report (IPAR) documenting this determination. Since such a determination requires the team to understand the system to a level at which such analysis can be made, evaluation evidence in the form of design documentation, training, informal vendor presentations, etc., is provided by the vendor to the evaluation team. This process typically begins when the vendor provides the evaluation team with product documentation (design, test, user, etc.) and system-level, developer-oriented training for the vendor's product. Training is followed by a comprehensive review of the system design by the evaluation team. The team performs security analysis of the product design. The team analyzes the system design and reviews the user and test documentation. The information gathered during design analysis is used to write an IPAR, which the team presents to a TRB. The team also briefs the TRB on the vendor's and the team's plans for testing the product.

After the IPAR/Test TRB meeting, the evaluation team performs security testing on the product. The test results are combined with the IPAR and edited to form the Final Evaluation Report (FER). The evaluation team presents final responses to open issues and the results of testing to a Final TRB. The TRB makes its recommendations to the TTAP Oversight Board regarding the product's requested trust rating. The Oversight Board then makes the final decision to place the evaluated product on the Evaluated Product's List (EPL) at the requested level. Finally, the EPL entry and the FER are published, and the evaluation documentation is archived.

1.2 Document Organization

This report consists of eight chapters, and three appendices. Chapter 1 is an introduction. Chapters 2 through 6 provide an overview of the system, its interaction with the operating system, software architecture, and a description of the security support (protection mechanisms and assurances). Chapters 7 and 8 provide a mapping between the requirements specified in the TDI and Interpreted TCSEC (ITCSEC) and the system features that fulfill those requirements.

Appendix A provides a list of acronyms. Appendix B provides a listing of the system tables in the SQL Server. Appendix C enumerates the components of the evaluated configuration.

2 System Overview

The Microsoft Corporation was founded in 1975 and has developed many commercial applications such as SQL Server, Excel, Word, Works, and PowerPoint. SQL Server 8.0 is a client/server relational database management system.

The evaluated configuration for SQL Server 8.0 includes one or more SQL Server instantiations installed on any number of both the Windows NT Server and the Windows NT Workstation products. The Windows NT products may act in any one of the following roles, connected via a network consisting of zero or more Windows NT domains:

Microsoft Windows NT 4.0 Server product
- Primary Domain Controller (PDC);
- Backup Domain Controller (BDC);
- Non-Domain Controller (domain member); and
- Non-Domain Controller (non-domain member).

Microsoft Windows NT 4.0 Workstation product
- Domain member; and
- Non-domain member.

The evaluated configuration supports replicated databases. This permits copies of whole or partial databases to exist on multiple SQL Server installations. However, the evaluated configuration does not support distributed databases. That is, data cannot be combined from different SQL Server installations to form a single database.

The evaluated configuration assumes that the physical network infrastructure (e.g., Ethernet) is protected and controlled by a single security administrative authority. This assumption is driven by the fact that a C2 evaluation does not generally address cryptography and other means of protection against unauthorized individuals that are able to gain physical access to the network media. Instead, the assumed scope of the C2 homogeneous network evaluation is the "system security architecture," assessing the ability of the SQL Server security architecture to protect resources from inappropriate access via the untrusted user and programming interfaces provided by each SQL Server product running on a Windows NT system node on the network.

There are a few commonly available SQL Server components that are not included in the evaluated configuration. Those components specifically not included in the 8.0 evaluation include:

- Distributed Transaction Coordinator;
- SQL Server Full-text Search;
- SQL Mail; and
- SQL Server Failover Support.

The next section will present an overview of the architecture of the SQL Server. The architecture will be explained in greater detail throughout the remainder of the report.

2.1 Architecture Overview

SQL Server is a database management system implemented using a client-server architecture. The SQL Server code that runs in the client and the code that runs in the server are the same and differ only in the installation options. Both SQL Server *client* and *server* configurations run on top of Windows NT 4.0 Service Pack 6a and C2 Update. This section provides an overview of the SQL Server architecture; the next chapter discusses SQL Server's interactions with Windows NT.

The client side of the SQL Server client-server architecture is part of the evaluated product because all administrative tools run on the client. Therefore the administrative tools, the libraries they use, and the protocols running in the client must work correctly to ensure that SQL Server administration performs as expected by the administrator. All security enforcement on the client side of SQL Server is provided by the underlying Windows NT 4.0 operating system. This security enforcement includes protection of the SQL Server executables and Dynamically Linked Libraries (DLLs).

The server side of the architecture, SQL Server, performs all security enforcing operations on behalf of applications running in the client. SQL Server runs on the server side as a trusted application with specific restricted permissions. When SQL Server is initialized, Windows NT creates an address space for SQL Server. Within that address space SQL Server executes as a process and performs its own memory management, managing the address space of requests for services from user client applications and external server requests. SQL Server requests and is assigned a pool of threads from Windows NT. Within this pool, SQL Server performs its own process management using the threads assigned to isolate individual task execution. SQL Server can also use fibers (light-weight threads), to perform process isolation between application requests whereby SQL Server does not require an operating system context switch to switch from one user context to another.

SQL Server relies upon Windows NT to perform user identification and authentication. For other security mechanisms including Discretionary Access Control (DAC) and audit, SQL Server provides its own support.

As described in this evaluation report, the SQL Server TCB includes the following system components:

- Open Data Services;
- Task Management;
- Memory Manager;
- Relational Engine;
- Storage Engine;
- Backup and Restore;
- SQL Server Agent; and
- Administrator tools.

The following sections provide an overview of the system components. Figure 2-1 System Architecture shows the system components.

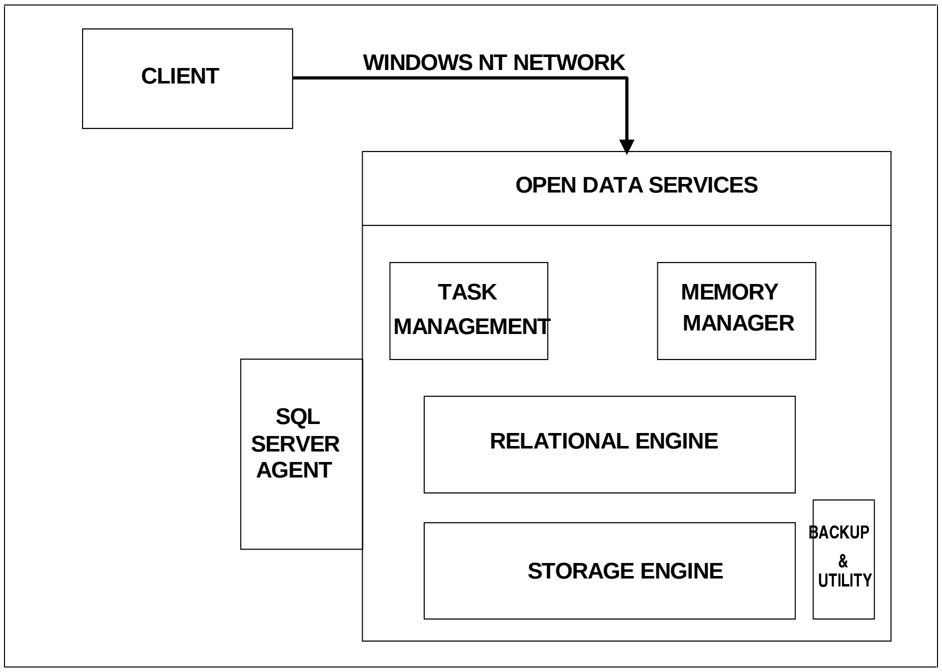

Figure 2-1 System Architecture

2.1.1 Open Data Services

The TCB interface for unprivileged programs accessing SQL Server is Open Data Services (ODS). ODS is the only interface between the SQL Server and clients performing requests. ODS processes Tabular Data Stream (TDS) packets to identify the type of packet and translate the packet type into a specific request type. TDS is an application-level protocol specific to SQL Server and is described more in the next section. All responses to user application requests return to the client through ODS.

2.1.2 Task Management

Task management is performed by the User Mode Scheduling (UMS) component that provides an OS-like environment for threads[1], including scheduling, and synchronization —all running in user mode, all (except for I/O) without calling the Windows NT kernel. UMS performs similarly to any process management within an operating system in that whenever a SQL Server component requires an execution context to be established, it calls UMS. UMS ensures separation of execution domains and performs execution scheduling. Client application requests participate in cooperative scheduling, voluntarily relinquishing control of the CPU so that the scheduler may process another execution context. There is no time-slicing or preemptive scheduling within UMS.

2.1.3 Memory Manger

The memory manager is responsible for the SQL Server memory pool. The memory pool is used to supply SQL Server with its memory while it is executing. Almost all data structures that use memory in SQL Server are allocated in the memory pool. The memory pool also provides resources for transaction logging and data buffers.

[1] Throughout the report the threads will be discussed as the unit of execution when either threads or fibers can be used.

2.1.4 Relational Engine

All security-relevant decisions are made in the relational engine. The relational engine establishes a user context, syntactically checks every TSQL (Transact SQL)[2] statement, compiles every statement, checks permissions to determine if the statement can be executed by the user associated with the request, optimizes the query request, builds and caches a query plan, and executes the statement. The main components of the relational engine are the event handlers, the query processor, the execution engine, and cursor support.

Each event handler calls a function within the relational engine and passes the appropriate elements of the TDS packet as arguments to the function. Similarly, when SQL Server needs to send information to the client, it calls the event handler which transforms the function call arguments into TDS events, and, using the callback, returns to ODS which builds TDS packet(s) for the event type with the information to be returned. ODS, in turn, sends the TDS packets to the client.

The query processor builds a query plan from TSQL statements. TSQL statements are parsed, optimized and prepared for execution. Within the query processor access permissions to objects are checked before optimization takes place. The output of the query processor, the query plan, is always placed in the procedure cache.

The execution engine executes a query plan from the procedure cache. The execution engine acts upon SQL server objects by requesting them from the storage engine. Once processing for a SQL statement is complete, the query processor returns the result to the appropriate event handler.

Cursor support exports an interface to allow the responses to client request to be formatted for windows with full cursor support.

2.1.5 Storage Engine

The storage engine is a resource provider. When the relational engine attempts to execute a TSQL statement that accesses an object for the first time, it calls upon the storage engine to retrieve the object, put it into memory and return a pointer to the execution engine. To perform these tasks, the storage engine manages the physical resources for SQL Server by making Windows NT kernel calls. When the relational engine (typically the execution engine) calls the storage engine to perform I/O, all access checks have been completed and the storage engine acts as an I/O processor. The main components of the storage engine are the access methods, page manager, transaction manager, lock manager, replication manager, log manager, and buffer manager.

The access methods and buffer manager provide in-memory support for database objects. The page manager supports retrieving database objects from disk.

The transaction manager works with the lock and log managers to control and log database transactions. The lock manager ensures transactional consistency. The log manager provides interfaces to three different types of logs: the transaction log, error log and the job history log. The log manager provides interfaces to set up, view, clear, display, resize, write, and delete the appropriate log and to backup the current transaction log.

Every SQL Server database has a transaction log that records all transactions and the modifications made by the transactions in the database. SQL Server exports an interface into the transaction log for the recovery of individual transactions, recovery of all incomplete transactions when SQL Server is started, and rolling a restored database forward to the point of failure.

The replication agents support replicated database information across SQL Server installations. The replication agents transfer updates between the copies of the databases.

[2] Transact SQL is the version of SQL that SQL Server uses for query processing. It is compliant with the SQL-92 standard.

2.1.6 Backup and Utilities

SQL Server utilities are a set of routines that typically support two functional areas: the Database Consistency Checker (DBCC) and backup.

The DBCC commands check the logical and physical consistency of a database, check memory usage, decrease the size of a database, and check performance statistics. Backup and restore utilities provide the capability to backup and restore a database. An administrator, the database owner or a user with appropriate authorization can create a database backup. A database can either be restored with the same name as an existing database or restored as a new database

2.1.7 SQL Server Agent

SQL Server Agent is a separate executable program that provides interfaces to create jobs and alerts. Jobs are a set of actions to be taken at a scheduled time. Alerts are actions to take when a specific event occurs (e.g., notify a person identified in the job when some action has occurred).

2.1.8 Administrator Tools

SQL Server provides several administration tools. These tools can be used to manage all aspects of the SQL Server including logins, access permissions, queries, and client connections. The administrator tools run on the client and are supported by several network libraries. These libraries are discussed in the next section. See Section 4.8 Administrator Tools on page 39 for a complete list of the administrator tools.

2.2 TCB Interface

An untrusted user requests services from the TCB through the TCB interface. The untrusted user interface for SQL Server is Open Data Services (ODS). Client applications are untrusted when they are executing at the request of an authenticated user who is not a SQL Server administrator, and they are running from an account without special Windows NT permissions. If any untrusted client software fails while making a SQL Server request, the client application may not provide anticipated results, but security cannot be violated. Since the network connecting the client and the server is a closed network supported by previously evaluated components, the connection between the client and the server is trusted; therefore the untrusted TCB interface is ODS running in SQL Server. It provides the only interface into SQL Server.

The administrative interface into the TCB is via the administrative tools that run on the client, because they must work correctly to maintain the security policy established for SQL Server. Since the administrative interface is a client interface, communication software between administrative client tools and the server are part of the TCB. This software includes ODBC Library routines to support the administrator tools: SQL Enterprise Manager, Query Analyzer, and Client Network Utility; Net-Libraries in both the client and the server; and the TDS protocol.

Figure 2-2 TCB Interface identifies the flow of client requests into SQL Server including the untrusted user interface as well as the administrative TCB interface. Windows NT Transport Driver Interface (TDI) and socket API (Winsock) are required to connect clients, running the SQL Server ODBC driver to the server on which SQL Server resides. The Named Pipes and Multi-protocol Net-library DLLs both support multiple network protocols (NW Link IPX/SPX, NetBEUI, and TCP/IP), however only TCP/IP is supported in the evaluated product.

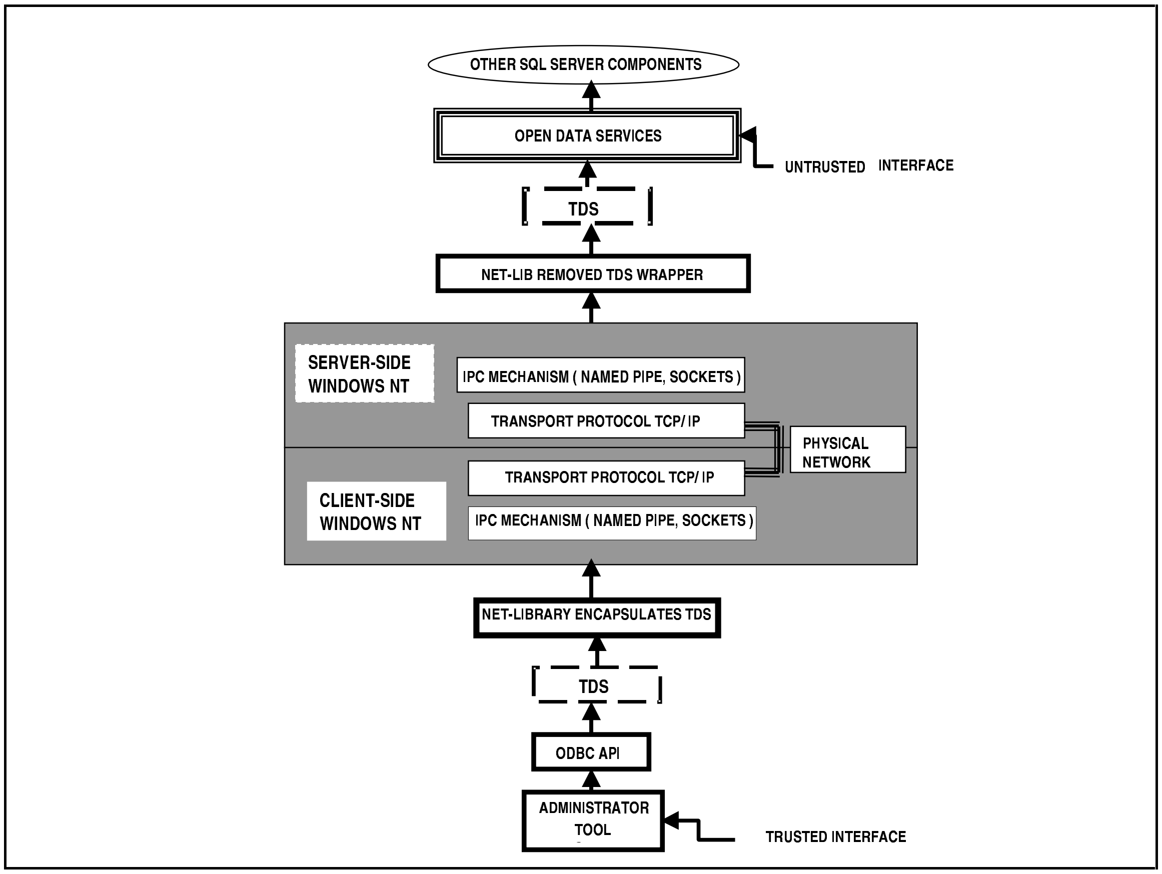
OTHER SQL SERVER COMPONENTS
OPEN DATA SERVICES
UNTRUSTED INTERFACE
TDS
NET-LIB REMOVED TDS WRAPPER
SERVER-SIDE WINDOWS NT
IPC MECHANISM (NAMED PIPE, SOCKETS)
TRANSPORT PROTOCOL TCP/ IP
PHYSICAL NETWORK
TRANSPORT PROTOCOL TCP/ IP
CLIENT-SIDE WINDOWS NT
IPC MECHANISM (NAMED PIPE, SOCKETS)
NET-LIBRARY ENCAPSULATES TDS
TDS
ODBC API
ADMINISTRATOR TOOL
TRUSTED INTERFACE

Figure 2-2 TCB Interface

2.2.1 Untrusted Interface

This section describes the untrusted user interface. It first describes ODS and then discusses Net-lib and TDS. Net-Lib and TDS are not the untrusted interface but their descriptions are necessary to understand how clients interact with SQL Server.

2.2.1.1 ODS

Untrusted clients connect into the server through an interface called Open Data Services (ODS). ODS provides a well-defined interface between server Net-Libs and server-based applications. It exports an interface consisting of function calls and macros used to develop ODS Server applications.

When requesting server services through ODS, client applications identify specific request events (e.g. login event, language event, attention event). These event types are translated into packet types and communicated to the server. Within the server, ODS recognizes these packet types and translates them back into events, whereby ODS calls a specific event handler. This type of client/server interface is not limited to the implementation of SQL Server, however SQL Server is implemented as an ODS application which means that SQL Server follows the client/server paradigm provided by ODS. This process is reversed when results are returned to the client.

Communications between SQL Server and its clients through ODS follow this sequence:

1. The server Net-Library receives TDS packets from the client.
2. The server Net-Library passes all packets directly to ODS.

3. ODS determines the type of request. If the request is to login or to open a database, ODS creates a SRV_PROC structure [3]and establishes a connection.
4. Requests coming from the client to the SQL Server and associated results are communicated over the established connection.

Every TDS protocol packet has an event identifier. For all TDS packets, ODS translates the TDS packet identifier into an *event_type* that it passes, using a callback interface, to server applications called an *event handlers*. Event handlers then use callback functions to send replies back through ODS to the client. Each type of packet maps to one event_type. All events are either server-level events or user-level events. The event types are briefly described in the following table.

EVENT TYPE	DESCRIPTION
ATTENTION	A client sends an interrupt request, or the network connection to the client is broken.
CONNECT	A client sends a connect request.
DISCONNECT	A client sends a disconnect request, or SQL Server calls an internal function to initiate a request to disconnect the current client thread.
EXIT	ODS calls a function to initiate a request for immediate shutdown.
LANGUAGE	A client sends a SQL Server language request. These requests manipulate database objects such as database, tables, view, and stored procedures. The language event is also used to load/unload large quantities of data into/out of databases. There are extended TSQL commands that invoke Windows NT directly, but those commands are restricted to administrative access.
RESTART	ODS calls a SQL Server function to continue a paused SQL Server.
SLEEP	ODS calls a SQL Server function to pause a started SQL Server.
START	ODS calls a SQL Server function to initiate the server.
STOP	ODS calls a SQL Server function to initiate a request to terminate all client threads and then shut down.

Table 2-1 ODS Supported Events

Server-level events (i.e. exit, start, stop, sleep, and restart) are restricted to administrators. Each server-level event has an individual event_handler to handle the specific request with a call to a specific function that is protected by the DAC policy, since each server-level event_handler first checks to see if the user is the administrator (sysadmin). All other events are handled by a user-level event handler (attention, connect, disconnect, language, login, logout).

To deal with simultaneous client requests, ODS creates a separate execution domain for each connection. When ODS receives a login or open database request, a separate connection is created by first allocating a separate thread for execution, then allocating a separate area in memory used for the process state, transaction descriptors, session descriptors, protection cache, login record, and parser work area. A separate `SRV_PROC` structure is created to contain connection information for each new client connection, and finally UMS is called and given a `SRV_PROC` pointer. When UMS returns, a per-session state associated with a specific connection has been established.

A new connection/session[4] is created for every Windows NT user running a specific application, from an individual host, over a specific port. A single user may have multiple concurrent connections active, but not from the same host, running the same port.

[3] There is one SRV_PROC structure per connection
[4] Connection and session are synonymous

Table 2-2 SRV_PROC Structure identifies the information maintained in the SRV_PROC structure for each connection.

SRV_APPLNAME	The application name supplied by the client when it logged in to the Open Data Services server application.
SRV_BCPFLAG	A flag that is TRUE if the client is preparing for a bulk copy operation, otherwise FALSE.
SRV_CLIB	The name of the library that enables the client to talk to a server.
SRV_CPID	The client process ID on the client's source computer.
SRV_EVENT	The current event for *srvproc*.
SRV_HOST	The host name used by the *srvproc* client application when it logged in to the Open Data Services server application.
SRV_LIBVERS	The version of the client library.
SRV_NETWORK_MODULE	The name of the Net-Library DLL used for the current *srvproc* connection.
SRV_NETWORK_VERSION	The version of the Net-Library DLL used for the current *srvproc* connection.
SRV_NETWORK_CONNECTION	The connection string passed to the Net-Library DLL used for the current *srvproc* connection.
SRV_PIPEHANDLE	A string containing the pipe handle of a connected client, or NULL if the client is connected on a network that does not use named pipes.
SRV_RMTSERVER	The server from which the client process is logged in. If the login is from a client, this value is an empty string.
SRV_ROWSENT	The number of rows already sent by *srvproc* for the current set of results.
SRV_SPID	The server thread ID of the *srvproc*.
SRV_SPROC_CODEPAGE	Codepage that the server uses to interpret multi-byte data.
SRV_STATUS	The current status of *srvproc*.
SRV_TDS	The version of the tabular data stream (TDS) used by the client
SRV_TYPE	The connection type of *srvproc*. If "server" is returned, *srvproc* is from a SQL Server. If "client" is returned, *srvproc* is from a DB-Library or ODBC client.
SRV_USER	The username used by the *srvproc* client application when it logged in to the Open Data Services server application.

Table 2-2 SRV_PROC Structure

2.2.1.2 Net-Lib

The Net-Lib architecture provides a method of sending TDS packets via an IPC mechanism across a physical network connection. Three Net-Libs are supported in the evaluated configuration, TCP/IP Sockets, named pipes, and multi-protocol. Multi-protocol Net-lib supports TCP/IP Sockets and named pipes. Net-Libs are implemented as DLLs, and multiple Net-Libs can be loaded simultaneously. A matching pair of Net-Libs must be active on the Windows NT client and server to support the desired network protocol.

The primary purpose of Net Lib is to manage user connections and interface with Windows NT. Net Lib is a connection management utility. It maintains the Windows NT handles for accessing the sockets associated with user connections. Net Lib performs no buffering of data. It does not understand the TDS protocol. It simply receives packets from Windows NT, removes the TCP/IP wrapper and forwards them to SQL Server. Any reassembly that may be necessary is handled by ODS. Similarly, when sending traffic, Net Lib adds a TCP/IP wrapper and forwards the packets to Windows NT.

Net Lib handles all connections for SQL Server. When Net Lib receives a connection from Windows NT, it puts the handle associated with the connection in an internal structure. The handle is for a socket pair that uniquely identifies a connection and is managed by the Windows NT kernel (see **the Windows NT 4.0 Service Pack 6a Final Evaluation Report** for a description of handles and sockets). Net Lib passes a pointer to the handle to SQL Server. The pointer is passed via a function call (`ConnectionAccept`) between Net Lib and SQL Server. SQL Server stores this handle in the `SRV_PROC` structure associated with the connection. Whenever SQL Server sends and receives data it includes the pointer into the Net Lib structure that manages user connections. Net Lib uses the referenced handle when sending data over a socket; thus ensuring the separation of connections.

2.2.1.3 TDS Protocol

Tabular Data Stream (TDS) is a message oriented application level protocol used for the transfer of requests and responses between client applications and SQL Server. Requests and responses are communicated through TDS packets. TDS packets are built by the client and then passed to a client Net-Library, which encapsulates the TDS packets into TCP/IP protocol packets. On the server, a server Net-Library that extracts the TDS packet and passes it to ODS receives the network protocol packets. The remainder of this section discusses the details of the TDS protocol.

At the client side interface, TDS data is one of the following ODS event formats:

- Login record (connect event);
- TSQL command (language event);
- TSQL command followed by its associated binary data (e.g., the data for a bulk copy command – language event);
- Attention signal (attention event); or
- Logout (disconnect event).

To send a TSQL command, or batch of TSQL commands, the TSQL command, represented by an ASCII string, is simply copied into the data section of a buffer and then sent to the SQL Server. A TSQL batch may span more than one buffer.

The client can interrupt and cancel the current command by sending an attention signal. Once the client sends an attention signal, the client reads until it gets an attention acknowledgment.

To send a logout request from the client, either the user enters LOGOUT at the command line or the application executes a disconnect. Both are received by ODS as a disconnect request and ODS calls the logout event handler.

2.2.2 Trusted Interface

The trusted user interface is composed of three portions: ODBC, Net-Lib, and TDS. Open Database Connectivity (ODBC) is the interface used by SQL Enterprise Manager, Query Analyzer, Profiler, and Client Network Utility running in a Windows NT client to administer SQL Server. ODBC is a Call-Level Interface (CLI) that allows these tools to access data from ODBC data sources, which for SQL Server is ODS. A CLI is an API consisting of functions administrative tools call to obtain a set of services.

The ODBC architecture four functional components are described in Table 2-3 ODBC Functional Components below.

COMPONENT	FUNCTION
Application	Calls ODBC functions to communicate with an ODBC data source, submits SQL statements, and processes result sets.
Driver Manager	Manages communication between the client application and the SQL Server ODBC driver.
Driver	Processes all ODBC function calls from the application, connects to SQL Server, passes SQL statements from the

	application to SQL Server, and returns results to the client application.
SQL Server ODS	Provides an API that presents all information a driver needs to access a specific instance of data in SQL Server.

Table 2-3 ODBC Functional Components

Administrative tools (trusted client applications) use an ODBC driver to access SQL Server data. The flow from a client application request to an event handler in the relational engine of SQL Server is briefly described. First, the trusted client application makes calls to the ODBC API using SQL statements written in either ODBC SQL syntax or SQL Server TSQL syntax (e.g., DBCC commands).

The client ODBC driver manager provides a standard driver interface for applications while allowing for driver changes. The driver manager primarily loads the modules comprising the driver and then passes all ODBC requests to the driver.

The SQL Server ODBC driver responds to all calls the application makes to the ODBC API. If the TSQL statements from the application contain ANSI or ODBC SQL syntax that is not supported by SQL Server, the driver translates the statements into TSQL syntax and then passes the statement to the server. The driver also presents all results back to the application. The driver communicates with the server through the SQL Server Net-Libs using the TDS protocol.

2.3 System Tables and Databases

The information used by SQL Server and its components is stored in special tables known as system tables. These tables are stored in system databases. The following sections describe the tables and databases.

2.3.1 System Tables

Successful operation of SQL Server depends on the integrity of information in the system tables. Direct updating, deleting, or inserting data in a system table can cause unpredictable effects. Therefore, the Trusted Facility Manual (TFM) instructs the administrator to prohibit direct updates to these tables. System tables are protected from modification by DAC. They are stored under the MSSQL directory, to which unprivileged users do not have write access. Also, Microsoft does not support triggers defined on the system tables.

Only SQL Server modifies the system tables in response to issued commands issued. Applications use the following components to retrieve information stored in the system tables:

- Information schema views;
- Stored procedures;
- TSQL statements and functions; and
- Database application programming interfaces (API) catalog functions.

System tables fall into two categories: those that contain global information and those that contain information specific to a single database within the server. The following are general descriptions of the contents of those tables. More detail is available in Appendix B on page 72. The descriptions in that Appendix are broken down into seven tables:

- Table B-1 Master Database Tables describes server-level system information tables that exist only in the `master` database;
- Table B-2 SQL Server Agent Tables describes server agent tables that exist only in the `msdb` Database;
- Table B-3 Database Backup and Restore Tables describes database backup and restore tables that exist only in the `msdb` database;
- Table B-4 Replication Tables in Master Database describes replication related tables that exist only in the `master` database;

- Table B-5 Replication Tables in Distribution Database describes replication tables that reside in the distribution database;
- Table B-6 Database System Tables describes database system tables that reside within every database; and
- Table B-7 Replication Tables in User's Database describes replication tables that reside within the user's database.

Some system tables are global across a SQL Server installation and some are local to a database created under SQL Server. Table 2-4 Global SQL Server System Tables and Table 2-5 Database Local System Tables indicate the global or local placement of the system tables. Details are not provided for each system table. As each table is referenced in the report, it is described in context.

In general, the Global System Tables are of the following types:

- Master Database Tables storing server-level system information.
- Server Agent Tables in the msdb database containing data used by SQL server agents. Agents perform functions associated with replication and routine administrative functions, such as backup/restore.
- Database Backup and Restore Tables that are stored in the `msdb` database. These tables store information used by database backup and restore operations.
- Tables used by replication that are stored in the master database, the distribution database, and the user's database.

Database Local System Tables consist of metadata describing the objects in a SQL Server database and are stored in every database.

sysaltfiles	sysjobhistory	sysdatabases	Mspublication_access
syscacheobjects	sysjobs	sysobjects	Mspublications
syscharsets	sysjobschedules	syspublications	Mspublisher_databases
sysconfigures	sysjobservers	sysreplicationalerts	Msreplication_objects
syscurconfigs	sysjobsteps	sysservers	Msreplication_subscriptions
sysdatabases	sysnotifications	syssubscriptions	MSrepl_commands
sysdevices	sysoperators	MSagent_parameters	MSrepl_errors
syslanguages	systargetservergroupmembers	MSagent_profiles	Msrepl_originators
syslockinfo	systargetservergroups	Msarticles	MSrepl_transactions
syslogins	systargetservers	Msdistpublishers	MSrepl_version
sysmessages	systaskids	Msdistributiondbs	Mssnapshot_agents
sysoledbusers	backupfile	Msdistribution_agents	Mssnapshot_history
sysperfinfo	backupmediafamily	Msdistribution_history	Mssubscriber_info
sysprocesses	backupmediaset	Msdistributor	Mssubscriber_schedule
sysremotelogins	backupset	Mslogreader_agents	Mssubscriptions
sysservers	restorefile	Mslogreader_history	Mssubscription_properties
sysalerts	restorefilegroup	MSmerge_agents	
syscategories	restorehistory	MSmerge_history	
sysdownloadlist	sysarticles	MSmerge_subscriptions	

Table 2-4 Global SQL Server System Tables

sysallocations	Sysindexkeys	MSmerge_genhistory
syscolumns	Sysmembers	MSmerge_replinfo
syscomments	Sysobjects	MSmerge_tombstone
sysconstraints	Syspermissions	sysarticleupdates
sysdepends	sysprotects	sysmergearticles
sysfilegroups	sysreferences	sysmergepublications

sysfiles	systypes	sysmergeschemachange
sysforeignkeys	sysusers	sysmergesubscriptions
sysfulltextcatalogs	MSmerge_contents	sysmergesubsetfilters
sysindexes	MSmerge_delete_conflicts	

Table 2-5 Database Local System Tables

2.3.2 System Databases

A SQL Server system contains four system databases, as described in Table 2-6 System Databases. These databases are used to maintain system information and temporary storage.

SYSTEM DATABASE	DESCRIPTION
`master`	The `master` database contains information on login accounts, system configuration settings, the existence of other databases, and initialization information for user databases and for SQL Server itself.
`tempdb`	The `tempdb` database provides server global temporary storage for temporary tables, stored procedures, and system worktables. . The `tempdb` is accessible to all users as a scratch area. It is treated as any other database except that it does not persist from one SQL Server session to another.
`model`	The `model` database is used as an initialization template for all databases created by SQL Server.
`msdb`	The `msdb` database supports SQL Server Agent in scheduling alerts and jobs, and in recording operators.

Table 2-6 System Databases

3 Composite Trusted Computing Base

The composite Trusted Computing Base (TCB) includes the SQL Server and the Windows NT 4.0 Operating System with Service Pack 6a and C2 Update. The evaluated configuration is one or more instantiations of SQL Server running in a client-server environment on any number of Windows NT Servers or Workstations. SQL Server relies on the operating system to enforce its security policy and to provide SQL Server with services and resources to implement the DBMS and its objects. Figure 3-1 Composite TCB illustrates the relationship of the operating system to the DBMS. The remainder of this chapter provides an overview of SQL Server database-specific terminology and describes the security policy and architecture of the composite TCB. Additional information about the Windows NT operating system can be found in the Windows NT 4.0 with Service Pack 6a and C2 Update Final Evaluation Report.

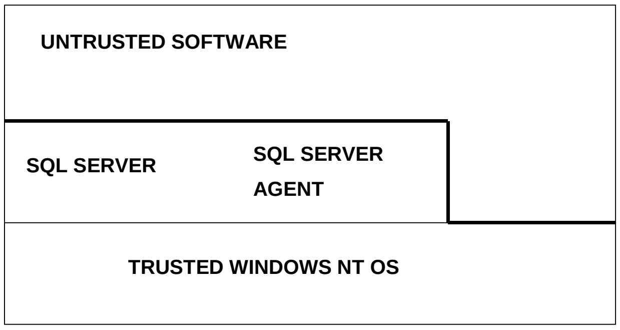

Figure 3-1 Composite TCB

3.1 Server Database Management System Concepts

SQL Server is a relational database management system (DBMS). The DBMS is responsible for enforcing the database structure, including:

- Maintaining the relationships between data in the database;
- Ensuring that data is stored correctly, and that the rules defining data relationships are not violated; and
- Recovering all data to a point of known consistency in case of system failures.

In a relational database, data is collected into tables that represent classes of objects that are important to an organization. Each table comprises columns and rows. Each column represents some attribute of the object represented by the table. Each row represents an instance of the object. Values in tables can be interpreted as pointing to like values in the columns of other tables, thus providing linkages among tables. When organizing data into tables there are many different ways to define the contents of a table. Relational database theory defines a process, normalization, that ensures that the set of tables defined will organize data efficiently and effectively.

Tables have properties that are an important part of the support for ensuring the integrity of data in a database. Data integrity refers to each occurrence of a column having a correct data value. The data values must be of the right data type and one of the allowable values.

The concept of referential integrity indicates that the relationships between tables must be properly maintained. Data in one table should only point to existing rows in another table; it should not point to rows that do not exist. Objects used to maintain both data and referential integrity include:

- Constraints;
- Rules;
- Defaults; and
- Triggers.

Constraints define rules regarding the values allowed in columns and are the standard mechanism for enforcing integrity. Rules specify acceptable values for a column. Rules are created separately from columns and can be bound to one or more columns. Defaults specify what values are used in a column if a value for the column is not specified when inserting a row. A stored procedure is a group of TSQL statements compiled into a single execution plan. Triggers are a special class of stored procedure defined to execute automatically when an UPDATE, INSERT, or DELETE statement is issued against a table.

To work with data within SQL Server, users issue a set of commands and language statements compatible with the DBMS software. The language used by SQL Server is Transact-SQL (TSQL), which supports standard SQL. Both the American National Standards Institute (ANSI) and the International Standards Organization (ISO) have defined standards for SQL. SQL Server supports the Entry Level of SQL-92, the latest SQL standard (published in 1992).

3.2 Composite Trusted Computing Base Security Policy

The composite TCB enforces a single security policy that restricts users from accessing information for which they have not been granted authorization. Information in the TCB is stored in objects. Access to information contained in these objects is mediated by the composite TCB in response to each subject's request. Subjects in the Windows NT OS and in SQL Server are Windows NT processes. Subjects and objects are discussed in more detail in Chapter 5 on page 42. SQL Server uses the following Windows NT DLLs when interacting with the operating system:

- KERNEL32;
- ADVAPI32;
- RPCRT4;
- OLE32; and
- VERSION.

The SQL Server Discretionary Access Control (DAC) policy augments the OS DAC policy to apply to SQL Server DBMS objects. The SQL Server associates an Access Control List (ACL) with each object and performs mediation based on ACL entries. Identification and authentication are provided by the Windows NT Operating System. A user must logon to the OS prior to connecting to SQL Server. There are two sets of administrative roles in SQL Server, server roles and fixed database roles. SQL server roles are discussed in detail in Section 5.3 SQL Server Roles on page 45.

3.3 Composite TCB Architecture

The SQL Server runs on the C2 evaluated Windows NT OS with Service Pack 6a and C2 Update in the evaluated configuration. SQL Server depends on the OS to function correctly and provide an underlying architecture to protect database objects and support system functions. SQL Server consists of two trusted services, SQL Server and SQL Server Agent. SQL Server contains the main DBMS executable to perform database and security functions. SQL Server Agent runs as a separate executable to execute jobs. SQL Server Agent is discussed in more detail in Section 4.5 SQL Server Agent on page 37.

3.3.1 Operating System Privileges

The SQL Server runs as a service Windows NT platform. The service account under which SQL Server runs must have four operating system privileges in order to perform security-relevant functions. Those privileges are:

- AssignPrimaryTokenPrivilege;

- IncreaseQuotaPrivilege;
- TcbPrivilege; and
- ServiceLogonRight.

There are two reasons for the use of privilege by the SQL Server. The SQL Server Agent is a separate process that must perform a login into SQL Server. In order to perform a login, the SQL Server Agent must make some Windows NT calls that require privileges. The second reason for requiring a privilege is in order to run as a service on Windows NT. This requires a special logon right. The details of the privilege uses are provided in the list below.

1. AssignPrimaryToken - the SQL Server Agent uses this privilege when establishing the user context for SQL Server Agent logon. The SQL Server Agent must call the CreateProcessAsUser Windows NT call.

2. IncreaseQuota - the SQL Server Agent uses this privilege when establishing the user context for SQL Server Agent logon. The SQL Server Agent must call the CreateProcessAsUser Windows NT call.

3. Tcb - SQL Server uses this privilege as part of SQL Server Agent logon. The SQL Server Agent must call the LogonUser Windows NT call.

4. ServiceLogonRight - This privilege is used to run the SQL Server Service.

3.3.2 SQL Server Communications

SQL Server can either listen on a named pipe or a socket. If it listens on a named pipe, Windows NT DAC protects the named pipe so that only SQL Server can access the private pipe. When SQL Server listens on a socket, there must be no untrusted programs running on the same Windows NT Server or Workstation. This restriction is necessary since Windows NT does not permit non-administrative users to bind to sockets exclusively. This restriction ensures SQL Server communications are protected. By default, SQL Server listens on port 1433

3.4 Windows NT Security Terminology

Through this report, reference will be made to specific Windows NT security terms. This section introduces those terms that are used in the report.

A *domain* is a logical grouping of network servers and other computers that share common security and user account information. The domain account information is defined on a Windows NT Server configured as a *Primary Domain Controller* (PDC). A *Backup Domain Controller* (BDC) contains a copy of the account information maintained on the PDC.

Windows NT Server Directory Services provide security across multiple domains through *trust relationship*s. Trust relationships are established between domains so that they may share account information in order to set ACLs on resources.

In Windows NT, users and groups of users are represented by accounts. Windows NT supports three types of user accounts:

- *User Account*: Represents a single, human user. Information included within a user account includes user name, full name, description, password, logon hours, logon workstations, expiration date, home directory, profile, associated groups, account type, and account condition (e.g. disabled and password never expires);
- *Global Group Account*: Represents a list of user accounts grouped together under one account name. This type of account can only be created on a domain controller. It can only contain global user accounts from the same domain. It cannot contain local user accounts or global group accounts from the same domain or other domains; and

- *Local Group Account*: Represents a list of user or global group accounts from one or more domains grouped together under one account name. It can contain local user accounts, global user accounts from the same domain or trusted domains, and global group accounts from the same domain or trusted domain. It cannot contain other local group accounts.

Each account created for any of the three user account types has a unique identifier called a *Security Identifier* (SID). A SID representing a user account is used for access control and other security functions.

4 System Architecture

This chapter describes the architecture of the SQL Server. It first discusses SQL Server's memory and task management, followed by a discussion its storage and relational engines. The chapter concludes with descriptions of backup and restore, SQL Server Agent, system startup, and administrator tools.

4.1 Memory Management

The SQL Server address space is divided into two distinct logical memory portions. The first is for its executable code. This executable code includes the SQL Server executable as well as network libraries and connection management code. The second portion of the address space is available memory for SQL Server processing of user requests. Memory management in SQL Server occurs using a *memory pools*. The memory pools provide memory resources to those portions of the SQL Server requesting memory. The memory pools can be divided into five categories:

1. Buffer pool;
2. System data structures;
3. Connection context information;
4. Procedure cache; and
5. Logging caches.

The first and largest category is the buffer pool. The buffer pool is used by the storage engine to store data that has been read from and written to disk. When the storage engine receives a request to access data, the buffer manager checks its pool to determine if it already has the data in memory. If it does not, the buffer manager makes a call to the OS to get the data; however, if the data is in memory, the buffer manager can simply return the data to the caller. In determining when a buffer can be reused, SQL Server uses a lazywriter. The lazywriter is a worker thread making continuous sweeps through all the buffers in memory. This worker thread marks as reusable any pages not in use that have not been touched since the last sweep.

The next category of memory is system data structures. These structures are used to hold global information about the SQL Server such as locks and database descriptors. A similar category of memory contains client connection context information. Each connection context contains information about the user context, open objects, parameters for stored procedures and RPC, and cursor positioning information. Clients do not have an interface to directly access their associated memory. Both of these categories are allocated from the SQL Server `resource` structure. The `resource` structure is a global structure from which all other system data structures are allocated. Throughout the discussion of the SQL Server architecture, descriptions are provided of memory structures that are all allocated from the `resource` structure.

The procedure cache is the next category of memory. The procedure cache is used to store compiled execution plans. When the compiler receives a request, it first checks the procedure cache to determine if a compiled plan already exists. This saves the SQL Server effort in producing compiled plans. More detail about the procedure cache can be found in Section 4.4.2.3 Procedure Cache on page 35.

A set of caches for logging is the last category in the memory pool. Database transaction log pages, used to maintain database consistency, are stored in a separate set of caches from the buffer pool. Figure 4-1 SQL Server Address Space shows the allocation of memory within the SQL Server.

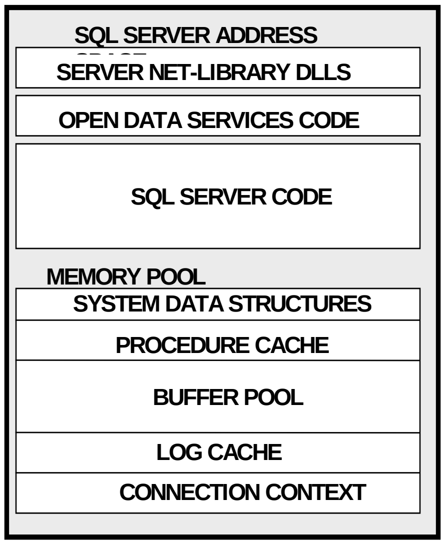

Figure 4-1 SQL Server Address Space

4.2 Task Management

SQL Server provides support for task management through the User Mode Scheduling (UMS) component. UMS is responsible for user and SQL Server system task scheduling and synchronization. This section describes the functionality and data structures for UMS in SQL Server.

4.2.1 Task Scheduling

UMS performs task management by managing a set of Windows NT process structures known as threads and fibers. In addition to managing Windows NT threads to represent processes, UMS can also be configured to manage Windows NT fibers. Fibers are Windows NT thread constructs that are manipulated via a set of Win32 APIs. Each thread can have multiple fibers. Fibers are subsets of threads contained within a thread object. They are scheduled by server applications without visibility to the operating system. Threads are allocated CPU time based on their thread priority, where fibers are not allocated CPU time by the system at all. Instead, the SQL Server application is responsible for scheduling fibers to run. A scheduled fiber runs only if the thread in which it is contained is scheduled to run. SQL Server operates in either fiber mode or thread mode, but not both simultaneously. Whether threads or fibers are used in scheduling is fixed for each running instance of SQL Server. Either thread or fiber process mode is effective until the server is restarted. The SQL Server configuration *lightweight pooling* option controls whether SQL Server uses threads or fibers and is represented in the `resource` data structure.

SQL Server thus maintains a pool of worker threads or fibers to perform task management. Threads or fibers and their associated data structures are used to represent user connections. SQL Server defaults to schedule a thread per concurrent user ODS event.

SQL server uses Windows NT threads or fibers to execute concurrent tasks on behalf of SQL server users and server system processes. UMS is responsible for creating one *UMS Scheduler object* (thread) for each CPU. If multiple schedulers are created, they run completely independently of each other. In fiber mode, UMS allocates one thread per CPU, then allocates a fiber per concurrent user command, up to the *max worker threads* server configuration option. In thread mode, UMS allocates threads from the worker pool of free threads. UMS uses the same algorithms to schedule and synchronize tasks when using either threads or fibers. Each user and SQL Server task has an associated data structure to store the thread/fiber user context.

UMS performs the following task management functions in SQL Server:

- Allocate Scheduler – Creates a new schedule and adds it to the system global list. Schedules run in fiber or thread mode;
- Delete Schedule – Waits for all users associated with scheduler to exit, then removes scheduler from global list and deletes structures;
- Create User (Thread/fiber) – Creates user structure and associates with a scheduler;
- Exit User (Thread/fiber) – Terminates user thread/fiber and releases associated data structures. Before terminating, the next user is chosen and account information is updated for the appropriate scheduler;
- Convert Thread – Converts currently running thread into UMS user including allocating context data structures and assigning a scheduler. Used during startup to move existing threads under UMS control.
- Suspend - Saves context for user and switches to next runnable user;
- Resume – Places a suspended user back on runnable queue for execution. User is placed in a ready state and continues execution based on previous context save point (saved through Suspend);
- Switch Preemptive – Moves user from non-preemptive to preemptive environment. Transfers control to NT thread outside of UMS control;
- Switch Non-Preemptive – Moves users back from preemptive to UMS non-preemptive environment.
- Event and Timer functions – Support for synchronization;
- Input/Output (I/O) functions – Support for performing and tracking user I/O requests. I/O cannot be performed the traditional way in the UMS environment because waiting for a synchronous transfer to complete would put the entire UMS environment to sleep. For this reason UMS provides functions to perform and track the I/O requests issued on behalf of users. All I/O processing and tracking is through the user's current scheduler. The UMS I/O mechanism allows one to issue any number of asynchronous I/Os, and then receive notice of their completion through completion routines; and
- Work Queue functions – Support for scheduler queues.

4.2.2 Task Execution

Each scheduler has an associated work queue for new requests. As new work arrives, the network thread places the requests on the appropriate scheduler's work queue based on where the UMS user is running. The work queues are populated from the working pool up to the specified limits as demand increases. If no idle workers are available as work enters the server, new workers are created to handle the load. Windows NT schedules threads/fibers for execution based on a numeric priority. UMS does not preempt running threads/fibers. A thread/fiber that does not voluntarily yield keeps processing. The time slice configuration option is used to prevent a looping thread/fiber from processing forever. Time slice defines how many milliseconds a process can execute without yielding. If a thread/fiber reaches the time slice value, UMS assumes it is stuck in a loop and terminates the task.

Each task has one of the following mutually exclusive associated states:

- Running – current task on scheduler;
- Suspended - waiting for resource or event;
- Waiting - Ready to Run; and
- Terminating.

UMS maintains a queue for waiting tasks ready to run. Other states are kept in data structures associated with the scheduler and the user task. Figure 4-2 Task State Transitions depicts the allowable state transitions.

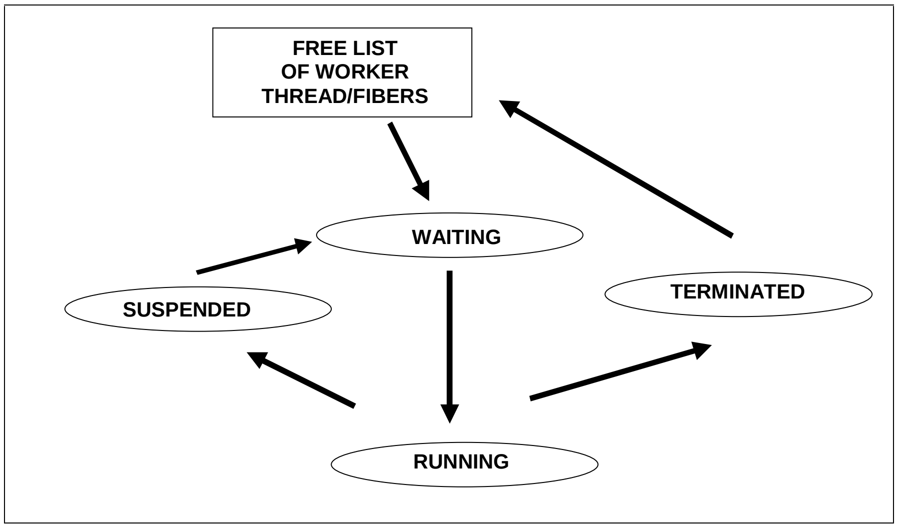

Figure 4-2 Task State Transitions

4.3 Storage Engine

The storage engine of the SQL Server controls access to system resources. It does not perform any access checks; rather it retrieves data for the relational engine. The storage engine's main functions are:

- Managing SQL Server data on disk;
- Transaction management;
- Locking;
- Replication;
- Logging and recovery; and
- Implementing utility functions such as the backup and restore.

Figure 4-4 Storage Engine shows the components of the storage engine. This section describes all the functions of the storage engine.

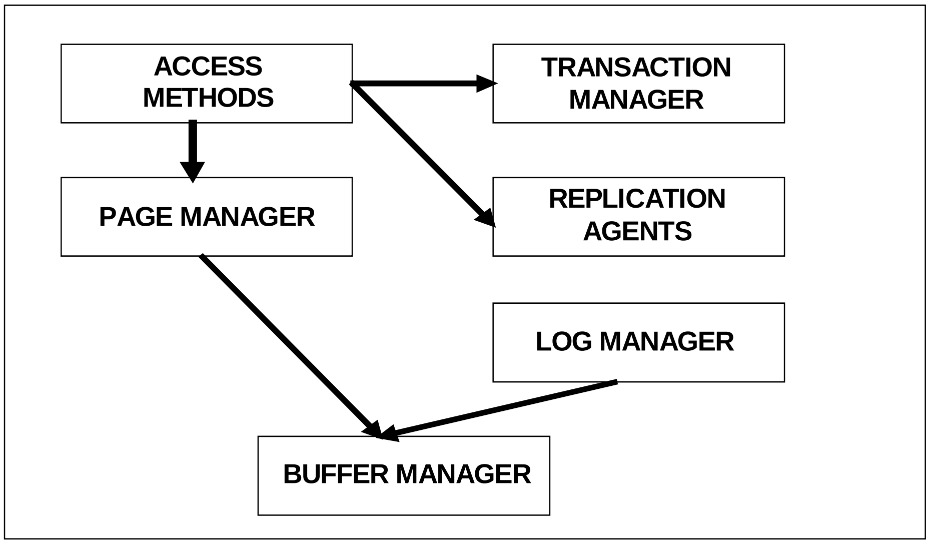

Figure 4-4 Storage Engine

4.3.1 Disk Management

SQL Server is responsible for managing its own data on disk. It stores all of its data in operating system files. Each database is stored in two or more files, with at least one for data and one for transaction log information. Data and transaction log information cannot reside in the same file and individual files can only contain information from one database. This section describes the disk management structures used by SQL Server to manage its data. It discusses files and file management, followed by a discussion of space management within the SQL Server.

4.3.1.1 Files and Filegroups

SQL Server uses operating system files to maintain its data. Those files can be grouped together into sets called filegroups for ease of allocation and administration. This section describes how SQL Server uses both files and filegroups.

4.3.1.1.1 Files

SQL Server is responsible for managing its files within Windows NT. Database files can be configured to grow automatically by user defined increments or can be statically sized. The SQL Server has three types of files it manages:

1. Primary – The first file of a database; it contains the starting location of all system tables associated with the database;
2. Secondary – These are all the data files for a database, excluding the primary file; and
3. Log – The log files contain the transaction log information needed to recover the database. There is a minimum of one file per database.

The first SQL Server page within each file is a file header page containing information about the attributes of the file. The ninth page in a primary data file is a database boot page containing information about the attributes of the database. Each file has a unique ID. Pages in a file are numbered sequentially starting at 0 for the first page in a file. Identifying a page within a file requires the file ID and page number. If a database only has two files, then it has a primary and log file.

Information about databases and their associated files is stored in two places. The `sysdatabases` system table in the master database contains one row for each database. Each entry contains the path for the primary file of a database. Within the primary file of a database there is a `sysfiles` table that has information about all the data and transaction log information that belongs to the database (i.e., associated files and filegroups).

4.3.1.1.2 Filegroups

Database files can be grouped together into *filegroups*. Filegroups are a named collection of one or more files that form a unit of allocation and administration. Files can only be a member of one filegroup. All data files can be part of a filegroup; however, log files cannot belong to a filegroup.

There are three types of filegroups:

1. Primary - The primary filegroup contains the primary data file and any other files not put into another filegroup. All pages for the system tables are allocated in the primary filegroup;
2. User-defined – User filegroups are those defined by a user at database creation or alteration; and
3. Default - The default filegroup contains the pages for all tables and indexes that do not have a filegroup specified when they are created. In each database, only one filegroup at a time can be the default filegroup. The primary filegroup is the default filegroup unless the user specifies otherwise.

Figure 4-5 Database Filegroups shows an example database that uses one primary filegroup and a secondary filegroup that contains two files. Note that filegroups may be stored on different drives. The `SysFileGroups` system table contains a description for all the filegroups in a database. There is at least one entry in this table for the default filegroup.

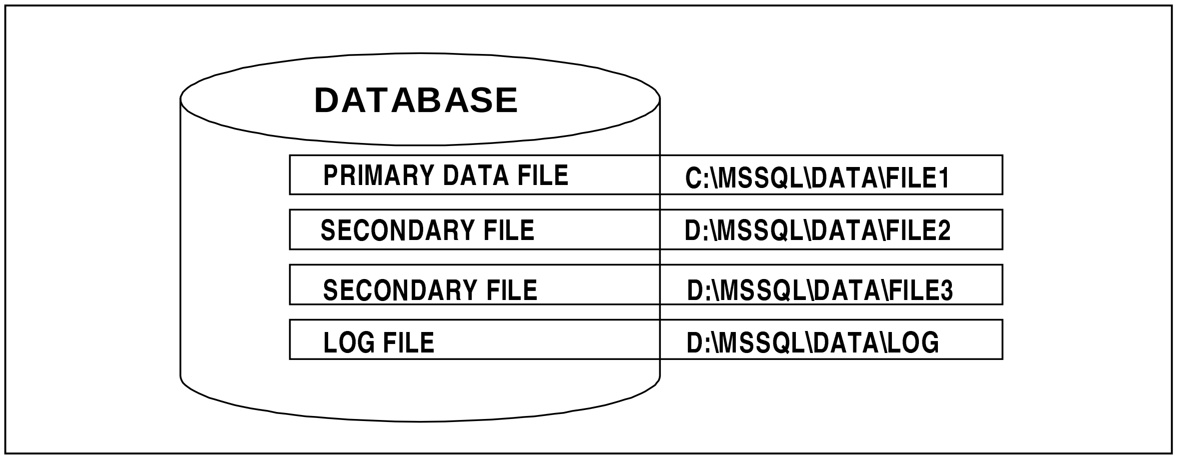

Figure 4-5 Database Filegroups

4.3.1.2 Page Management

The basic unit of data storage in the SQL Server is a *page,* which is 8K in size. When a database needs additional space, an extent is allocated. An extent is eight pages. Figure 4-6 Page Structure shows the layout of a page. The page header is 96 bytes describing the page. The header is followed by the body, which is populated with rows. The rows themselves contain system and user data. The slot array consists of entries that point to starting positions of rows on the page. The slot array begins at the end of the page and grows backward.

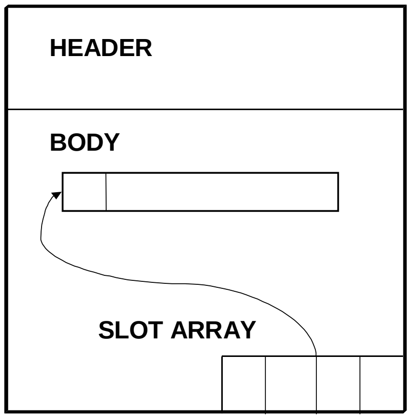

Figure 4-6 Page Structure

The page header is 96 bytes and contains two sections. The first section contains information about the page itself, including the page number, page type, and which object is associated with the page. Pages are either associated with tables or indexes. The second section contains information about the data including number of rows, amount of used space, and end of data. Rows within the body of a page contain a header and data. The header is used to specify the type of data in the row, e.g., variable length, index, ghost. The slot array is used to find rows within the page. Whenever a row is added to a page, an entry is added to the slot array specifying the index of the row within the page.

Rows are added to the end a page. As rows are deleted, space becomes available at the beginning of a page. To reclaim this space, SQL Server compacts the page. When a row is deleted it is marked as a *ghost* row. This means the row is no longer available to users but has not been cleared from the page. A background process runs and clears ghost rows from pages. After the ghost rows have been cleared (i.e., zeroed), the remaining rows are compacted at the beginning of the page and the slot array is updated. Rows can also be cleared upon an insertion. If contiguous space is not available upon insertion, the page will be compacted and rows can then be inserted at the end of the page.

4.3.1.3 Page Allocation

The Space Manager is responsible for managing pages within the SQL Server. Space management occurs on a file by file basis with each file's space management data structures being completely self-contained within a given file. Free space is managed in the following four ways:

1. Global Allocation Map (GAM) – GAM pages record the free uniform extents within a set of pages. Pages in a uniform extent belong to a single table or index;
2. Small Global Allocation Map (SGAM) - SGAM pages track the free mixed extents within a set of pages. Pages in mixed extents are shared among tables or indexes;
3. Page Free Space (PFS) – A PFS page tracks the free space within a set of pages. The number of pages managed by the PFS is a multiple of the extent size; and
4. Index Allocation Map (IAM) – These pages track extents used by a particular table or index.

The Space Manager has three subcomponents to control the type of pages just described. The three subcomponents and the types of pages they manage are:

1. GAM Manager – manages GAM pages
2. SGAM Manager – manages SGAM pages; and
3. Page Manager – manages PFS and IAM pages.

This section describes each subcomponent and its role in page management.

4.3.1.3.1 GAM Manager

The GAM Manager is responsible for managing uniform extents and GAM pages. GAM pages contain a bitmap representing free extents within a given page range. GAM pages occur at the beginning of an extent and are the first page of an extent unless a PFS page exists and then the GAM page is the second page. A GAM page contains two records – one with header information such as size and number of extents, and the second record with a bitmap with each bit representing an extent in the page range. If the bit for an extent is set, the extent is available for allocation and visa versa. Every extent in the page range has a corresponding bit in a GAM bitmap to represent its allocation state.

The Page Manager and SGAM Manager make requests to the GAM Manager to allocate or release an extent. Requests from the Page Manager are for a uniform extent. Conversely, requests from the SGAM Manager are for mixed extents. There is no interface to the GAM Manager for untrusted users.

During initialization of a GAM page, all of the bits in the bitmap are set indicating that the entire range of extents managed by this bitmap are free. When a request is made for a free extent, the GAM pages are scanned sequentially looking for bits that are set, the allocation is logged, the corresponding bit for the extent is cleared in the GAM, and the extent id is returned to the caller with the extent locked. When a request is made to free an allocated extent, the GAM page that manages that extent is computed, the GAM page is brought into cache, the deallocation is logged and the corresponding bit for the extent is set.

4.3.1.3.2 SGAM Manager

The SGAM Manager is responsible for managing mixed extents and SGAM pages. SGAM pages contain a bitmap indicating which extents have free pages available for use in a mixed extent. These pages occur at identical intervals in the device as the GAM pages but occupy the second page in the extent as the GAM page uses the first page. When the SGAM page falls on an extent that also has a PFS page, it occupies the third page of the extent.

The SGAM structure is identical to the GAM with the exception that its bitmaps represent extents that are currently supporting single page allocations and which have free pages available. It is initialized in the same fashion as the GAM with the difference that all of the bits in the bitmaps are cleared at device initialization time, indicating that no extents are currently supporting single page allocations.

The SGAM Manager only receives requests from the Page Manager for the purposes of allocating a single page. The SGAM Manager, in turn, calls the GAM Manager when it needs to transition an extent from uniform to mixed. Initially, all extents are uniform. If the SGAM Manager receives a request for a single page, it must call the GAM Manager to obtain an available extent.

4.3.1.3.3 Page Manager

The Page Manager is responsible for managing free space within a page, page allocation and deallocations, and extent allocations and deallocations. Additionally, it is responsible for managing PFS and IAM pages. PFS pages record whether a page has been allocated and the amount of free space on the page. Each PFS page maintains data for about 8,000 pages. After an extent has been allocated to an object, the Page Manager uses the PFS pages to record which pages in the extent are allocated or free and how much free space is available for use. This information is used when Page Manager has to allocate a new page, or when it needs to find a page with free space available to hold a newly inserted row.

IAM pages map the extents in a database file to a given object. Objects in the database are stored in either tables or indexes; hence, IAM pages track table and index allocation. Each table or index has one or more IAM pages recording all the extents allocated to the object. A table or index has at least one IAM for each file on which it has extents. A table or index may have more than one IAM on a file if the range of the extents for the table or index on the file exceeds the range that an IAM can record. By default, a mixed extent is used for the first eight pages of a table.

An IAM page has a header indicating the starting extent of the range of extents mapped by the IAM. The IAM also has a bitmap in which each bit represents one extent allocated to the object. If a bit is 0, the extent it represents is not allocated to the object owning the IAM. If the bit is 1, the extent it represents is allocated to the object owning the IAM page.

When the Page Manager needs to insert a new row and no space is available in the current page, it uses the IAM and PFS pages to find a page with enough space to hold the row. The Page Manager uses the IAM pages to find the extents allocated to the object. For each extent, the Page Manager searches the PFS pages to see if there is a page with enough free space to hold the row. The Page Manager allocates a new extent to an object only when it cannot find a page in an existing extent with enough space to hold the row being inserted. When allocating an extent, the Page Manager calls the SGAM or GAM Manger to service the request.

4.3.2 Access Methods

The Access Methods portion of the storage engine is responsible for the management of data and the manipulation of database constructs while in memory. The Access Methods maintain several data structures for managing database objects in memory. These data structures are allocated from the `resource` structure.

4.3.3 Transaction Management

A transaction is a sequence of operations performed as a single logical unit of work. The unit of work may be a single SQL statement or a series of SQL statements, which begin with a START transaction statement and continue through an END transaction statement. The logical unit of work must exhibit the following four properties:

1. *Atomicity* - a transaction is an atomic unit of work; either all of its data modifications are performed, or none of them are performed.

2. *Consistency* - when successfully completed, a transaction leaves all data in a consistent state and all internal data structures, such as b-tree indexes or doubly linked lists, must be correct at the end of the transaction. If unsuccessful, all of the data modifications made since the transaction started are undone.
3. *Isolation* - modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions. A transaction either sees data in the state it was in before another concurrent transaction modified it, or it sees the data after the second transaction has completed, but it does not see an intermediate state.
4. *Durability* - after a transaction has completed, its effects are permanently in place in the system. The modifications persist even in the event of a system failure.

4.3.3.1 Transaction Logging

SQL Server stores changes to the database in transaction log files. These files are a set of OS log files containing contiguous log records. New log records are appended to the end of the currently active log. When the file becomes full, log records roll over to a new log file. Once all the log files are full, log files are reused to store a new record. Log files belong to the SQL Server and are protected from unauthorized access by Windows NT DAC security. Figure 4-8 Transaction Log Representation shows an example transaction log.

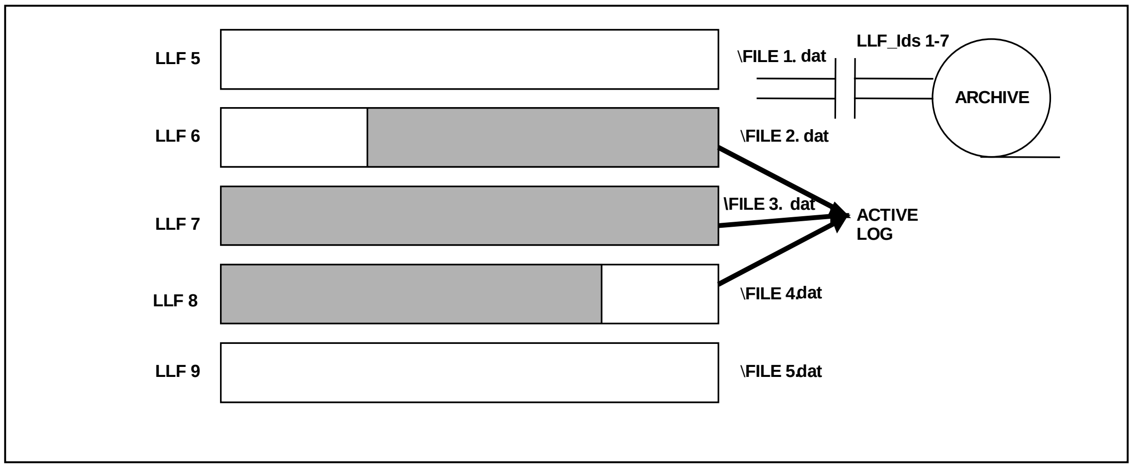

Figure 4-8 Transaction Log Representation

Figure 4-8 Transaction Log Representation represents log files which were filled up once and are about to be filled up a second time. `\file4.dat` is the current file.

Since a single file can contain multiple instances of the log, each instance of the log file is called a *logical log file* (LLF). A unique 4-byte number called LLF Identification (LLF_id) identifies the logical file. All log records contained in `\file1.dat` when it was filled the first time around form LLF 1. The log records which filled `\file1.dat` the second time around formed LLF 5 and so on.

Each record within the log is associated with a unique 8-byte integer called the Log Sequence Number (LSN) made up of two parts: a 4-byte LLF_id and a 4-byte offset within the LLF. The LSN of a log record is the LSN of the starting byte of the record. A log record is said to be before another if its LSN is smaller than the LSN of the other log record.

The shaded portion between Minimum LSN (MinLSN) and the end of the log is called the active log. This is the portion of the log used for recovery after a system crash. In the figure above, `\file1.dat` does not contain any portion of the active log and is marked "inactive" while the rest of the files are marked "active" as they contain some of the active log.

When a physical file becomes full and a switchover to another log file occurs, the old log file may be backed up to an archive as shown in Figure 4-8 Transaction Log Representation. Once the contents of the entire physical file is backed up, it is marked “backed-up” and “inactive” and can be reused to store another LLF. In the above picture, file `\file1.dat` is available to be reused to store LLF 9 when `\file4.dat` is full, but `\file2.dat` is not available for reuse since it is still "active".

To optimize writing log records, the log records are formatted in a region of memory called the log cache (LC). Once the LC becomes full or if a user issues a checkpoint command, the contents of the LC is written to LLF.

The log manager provides for both forward and backward scans of the log. In the backward scan, the log manager returns the log records for a particular transaction which are linked in the backward direction. The log manager reads each log record and finds the LSN of the previous log record from the current log record. During this backward scan, log manager uses large I/O blocks in an attempt to read in more than one log record in a single I/O. This kind of scan is used by rollback of a transaction. In the forward direction, the log manager starts at a given point and returns all the log records from that point to the end of the log. The replication log reader uses this kind of scan.

4.3.3.2 Lock Management

In a multi-user environment, there is a high probability that users will require the same resource during their threads. SQL Server assures that users will not interfere with one another by invoking a *lock manager* that places locks on resources. There are six different lock types as follows**:**

- *Exclusive* - prevents access to a resource by concurrent transactions and is released at the end of the transaction;
- *Shared* - allows concurrent transactions to read a resource but will not allow either transaction to modify the data;
- *Update*– prevents another resource to concurrently attempting to modify the same resource. This lock does not prevent concurrent transactions to read the resource;
- *Intent* - indicates that SQL Server wants to acquire a shared or exclusive lock on some resource lower down in the hierarchy. This removes the need to examine every row or page lock on the table to determine if a transaction can lock the entire table;
- *Schema* - occurs when a table data definition language (DDL) operation or schema modifications is being performed and prevents users from accessing the schema; and
- *Bulk update* – prevents access to a database table when bulk-copying data into the table and the *TABLOCK* parameter has been specified.

4.3.3.3 Resource Locking

The lock manager has multi-granular locking capability that allows different types of resources to be locked by a transaction. To minimize the cost of locking, SQL Server locks resources at a level appropriate to the task. The lock manager can lock these resources as follows (listed in order of increasing severity):

- *Row identifier* - used to individually lock a single row within a table;
- *Key* - a row lock within an index used to protect key ranges in serial transactions;
- *Page* – an 8 KB data page or index page;
- *Extent* - contiguous group of eight data pages or index pages;
- *Table* - entire table, including all data and indexes; and
- *DB* – entire database.

4.3.3.4 Spinlock

To allow mutually exclusive access to a resource, each thread requests ownership of the resource before executing a section of code that accesses the protected resource. When it has finished executing the protected code it relinquishes ownership, enabling another thread to become owner and access the protected resource. The object that effects the exclusion mechanism is *spinlock*. The *spin counter* option of SQL Server limits how many times a thread can loop on a lock before it temporarily sleeps. This allows other tasks to run. When the thread wakes up, it retries the spinlock.

4.3.3.5 Lock Escalation/De-escalation

Once a lock is acquired, processing may result in changing the lock granularity by either of the following processes:

- *Lock Escalation* –increasing the lock's severity which results in minimizing the number of locks that may be acquired by a transaction to avoid deadlock; and
- *Lock De-escalation* **-** minimize the locking by acquiring finer-grained locks on resources.

The lock manager scans the outstanding page locks to determine if the threshold of outstanding page locks on a table are more/less than has been configured. When the threshold is exceeded, the lock manager attempts to convert the table level intent lock into a more severe lock in order to release the underlying page locks.

The lock manager also performs a periodic deadlock detection check for waiting lock requests by scanning the thread list for threads waiting on a lock request. Each of these threads has a flag indicating it has been visited by the deadlock checker. If a waiting thread does not have this flag set, the deadlock manager will set the flag and move on. If the flag is set, the thread has been waiting on a lock for longer than the deadlock detection period and is considered for lock escalation. If there are no other active locks associated with a lock, the lock will be released or de-escalate.

4.3.4 Replication

The SQL Server supports transactional replication of data between different SQL Server instantiations. All or part of the data from a database is copied among SQL Servers. The evaluated configuration does not support the concept of distributed databases. Distributed databases permit a single database to span multiple SQL Servers. This section describes the replication architecture followed by a description of the replication access control policy.

4.3.4.1 Replication Architecture

Replicated data is originated on a *publisher's database* and is copied onto a *subscriber's database*. The SQL Server that creates the data to be copied is the publisher. The publisher writes data to a *distribution database* located on a *distributor.* The SQL Server that receives copies of the data is called a subscriber. There may be more than one publisher for a distribution database and there may be more than one subscriber. The remainder of this section describes how the SQL Server performs replication.

An agent called a log reader reads the publisher's transaction log. The log reader is responsible for reading the transaction log of the database to be replicated and moving new transactions marked for replication into a distribution database. The log reader agent runs at the distributor. There is one log reader agent per publisher serviced by the distributor. The distribution agent applies the transactions in the distribution database to a subscriber. Transactions are applied in the same order, providing transactional consistency with minimal latency in delivery to each subscriber. The log reader and distribution agents can be scheduled to run at regular intervals looking for any new transactions or can be executed on-demand.

Transactions are passed to subscribers in one of two ways – push and pull transactions. The distribution agent runs at the distributor for push subscriptions and at the subscriber for pull subscriptions. In the push case, a replication thread on the subscriber polls the distribution database looking for new transactions. The replication thread compares the transaction ID in the distribution database with the last transaction ID on the subscriber. New transactions are applied at each subscriber on a connection held by the distribution agent. In the pull case, the distributor sends information to the subscriber.

Replication requires the agents to have a valid login account and password when connecting to a publisher, distributor, or subscriber. The replication agents run under the security context of the SQL Server Agent service. There are three cases where logins occur:

1. The log reader agent connects to the publishing database at the publisher and to the distribution database at the distributor;
2. In a push subscription, the distribution agent is located on the distributor and connects first to the distribution database on the distributor. While connected to the distributor, the distribution agent connects to the subscribing database at the subscriber; and
3. In a pull subscription, the distribution agent is located on the subscriber and connects first to the subscribing database on the subscriber. While connected to the subscriber, the distribution agent connects to the distribution database at the distributor.

In additional to possessing a valid login, the distributor maintains a Publication Access List (PAL) that specifies which subscribers are permitted to access which publication data.

4.3.4.2 Replication Access Control

When replication is established, the administrator specifies an owner for the data on each publisher. By default the owner for the data on the remote subscriber is the same as the data on the local publisher. All access control decisions to the replicated data are managed with the local copy of the data. Permissions are stored in the system tables and system tables do not replicate.

The implication of this security paradigm is that the data on the publisher and the subscriber will most likely have different security permissions associated with it. The owner of the data on each publisher and subscriber gets to set the permissions on the associated copy of the data. As new data is fed into the subscribers, the subscriber owners, not the original owner, set the permissions. If the owner of the data wants to maintain full control of the data, the data should not be replicated. Auditing is performed on the local copy of the data.

4.4 Relational Engine

The relational engine comprises two main components and cursor support (see Figure 4-9 Relational Engine Architecture). The two main components are the event handlers and the query processor. This section discusses all of the components in the relational engine and their interaction with other components in SQL Server.

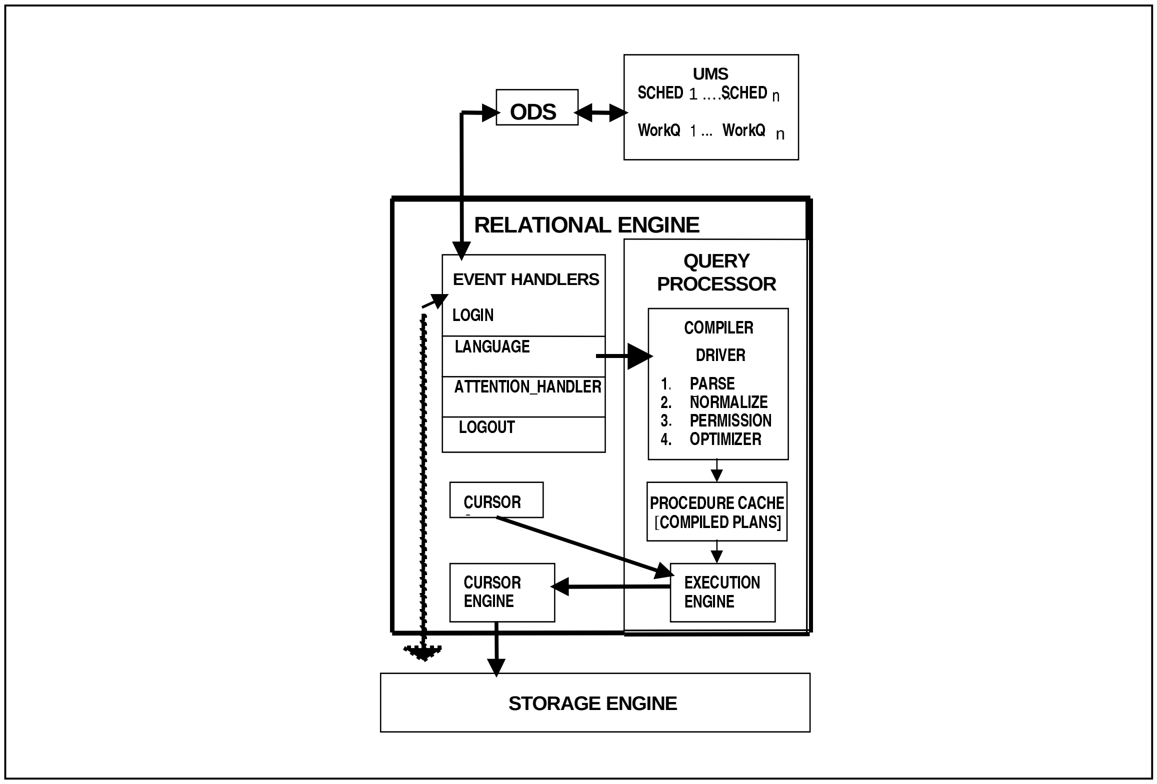

Figure 4-9 Relational Engine Architecture

4.4.1 Event Handlers

All requests from ODS to the relational engine are in the form of an ODS event. Every TDS protocol packet has an event identifier. For all TDS packets, ODS translates the TDS packet into an event_type and calls the relational engine's event handler for the specific event_type. Event handlers belong in one of two categories:

- Connection event handlers, which process client connects and disconnects; and
- Language event handlers, which process database language (TSQL) requests.

4.4.1.1 Server Events

Server events may be generated by ODS in SQL Server by calling one of two library routines and passing a parameter identifying the type of event. When SQL Server internally calls an event handler to process a server event, the event is not compiled or run from the procedure cache as are client events or server events generated from administrative software running on the client.

Server generated events are identified in the following table:

EVENT ACTIVATED	DESCRIPTION OF EVENT
DISCONNECT	Requests a disconnect for the current client thread.
EXIT	Requests an immediate shutdown.
SLEEP	Requests pausing a started server.
RESTART	Requests continuing a paused server.
STOP	Requests a graceful shutdown, first waiting for client threads to terminate and then shutting down.

Table 4-10 ODS Generated Events

4.4.1.2 Client Events

For all client events and server events generated by client tools, each event handler calls a function within the relational engine and passes the appropriate elements of the TDS packet as arguments to the function. Similarly, when SQL Server needs to send information to the client, it calls an event handler that transforms the function call arguments into events, which are encapsulated in TDS packets, and returns the event to the client.

The event handlers that service client event requests are the following:

- LOGIN;
- Language events;
- Attention events; and
- LOGOUT.

4.4.1.3 Login Event Handler

Login event handler is the handler that puts the server in a *running state*, that is no client request other than login will be allowed to pass `ODS` until the server is in this state. Login also enforces a "timeout" for login event packets that are not followed by subsequent packets within a specific time interval[5]. Login builds a `SVR_PROC` structure and places it on a time queue. If a subsequent packet does not arrive within the time interval, the login event handler returns a "disconnect" request to `ODS` and the connection is disconnected. When a subsequent packet arrives within the time interval, the login event handler removes `SVR_PROC` from the timer queue and performs the following:

- Asserts the server is in running state;
- Creates a login record in the `SVR_PROC` structure;
- Calls UMS to create a worker thread;
- Populates the `SVR_PROC` structure with the thread ID, a pointer to the user context, and free memory area;
- Builds the security cache to maintain security state;
- Calls a handler to process the event type identified in the subsequent packet; and
- Sends a login acknowledgement back to `ODS`.

4.4.1.4 Language Event Handler

The language event handler is called for all language events. For SQL server, the only language events are TSQL commands and batches of TSQL commands. This is the first event handler called to perform work for the client application, therefore the language event handler calls `UMS` to create a thread or fiber to process the TSQL transaction. The language event handler sends one entire transaction to the compiler. The results from the TSQL query are returned to this event handler, which in-turn returns the results to `ODS`.

[5] The login timeout is to protect against spurious packets arriving (e.g. line noise) that by chance appear to be a login packet.

4.4.1.5 Attention Event Handler

The attention event handler is called to process *attention signals*. Attention signals provide a way for client applications and SQL Server to flag and monitor potential problems and error conditions. Attention events are serviced as soon as they are detected.

An attention signal may come from `ODS` to the event handler, when a client-interrupt request or a broken client connection has occurred, or the event handler can be called from within SQL Server when servicing a request**.** The attention event handler also may be called when a client application sends an attention event to query for attention signals associated with the client connection.

4.4.1.6 Logout

The Logout event handler is called to disconnect the client application from the server. Logout is called when the user executing the client application enters a logout. The Logout event handler is also called when a client application executes an extended SQL (E-SQL) *disconnect* instruction or when the application exits. Each of these events creates a logout TDS packet type that `ODS` routes to the Logout event handler.

The Logout event handler is called by SQL Server when it wants to disconnect after a thread has exceeded its allocated inactive time. During the process of disconnecting, Logout calls the UMS for deallocating and clearing the execution context.

4.4.2 Query Processor

The query processor is composed of the compiler, the procedure cache, and the execution engine. The query processor syntactically checks every TSQL statement, compiles every statement, checks permissions to determine if the statement can be executed by the user associated with the request, builds and caches a query plan, and executes the statement. It is within the query processor that all access checks are made.

4.4.2.1 Compiler

The compiler is subdivided into the driver, parser, permission check, and the optimizer. The compiler first compiles every TSQL statement that is to be executed by SQL Server. SQL Server will accept no statements or procedures that could be directly executed without compilation by the compiler.

The process of building a query plan is called *compilation.* TSQL statement compilation occurs regardless of the source of the statement or the number of statements included in one transaction. Each transaction is compiled into one query plan. Because query plans are held only in the procedure cache and not on disk, they must be rebuilt each time SQL Server starts.

4.4.2.1.1 Driver

The driver determines if an execution plan for the client request already exists in the procedure cache (see Section 4.4.2.3 Procedure Cache on page 35). For every transaction of TSQL statement(s) to be executed by SQL Server, the driver first looks through the procedure cache to see if there is an existing execution plan for the same TSQL statement(s) with the same execution context. SQL Server reuses any existing plan it finds, saving the overhead of recompiling. If there is no existing execution plan, SQL Server continues the compilation. If the execution plan is in the procedure cache, access checks are performed and the *execution engine* is called to perform the statements.

4.4.2.1.2 Parser

The Parser performs the second phase of TSQL statement compilation. The Parser scans each TSQL statement, determines that the syntax is correct, and breaks it down into the logical units, such as keywords, parameters, operators, and identifiers. An entire transaction is parsed and if there is a syntax error in any one statement the processing ceases and an error is returned to the language event handler. If the syntax is correct, each TSQL statement is broken into steps and a query tree is created.

4.4.2.1.3 Permission Check

Once the parser has built the query tree, the compiler checks that the user associated with the application has sufficient permissions to perform the TSQL statements within the transaction. The permissions check in the compiler occurs before the query is optimized, since the optimization process is expensive. Permission to access the object and the type of access requested is determined by checking the user ID, the type of access requested by the user, the TSQL statement, and the permission identified in the `syspermissions` system table.

When the permission check fails, the compiler returns an error to the language event handler. The language event handler sends a numerical error value to `ODS`, which in turn looks in the `sysmessages` system table, retrieves the error message description and error number and returns this information to the client. Access violation errors only identify whether access was denied to a database, or to a log file, but no further granularity (e.g., table, and row) is reported. When a permission check is made while compiling a single batch of TSQL statements, the first permission failure returns an error and further compilation ceases.

Once the permission check has reported that the user has the appropriate permissions to perform the transaction, a timestamp, the identification of each object, and the permissions associated with the access check and the type of access granted are saved in a security cache that is associated with every user. This timestamp is critical to the enforcement of the DAC. After an entry is made in the security cache for each specific object, the TSQL statements associated with that object are optimized.

4.4.2.1.4 Optimizer

The final phase of compilation is the Optimizer. The Optimizer passes over the logical tree produced by the parser multiple times to select a least-cost execution plan. Once an execution plan is selected, the query is rewritten into semantically equivalent queries. The Optimizer also finds algorithmic implementations for each of these representations. Lastly, using one of these implementations, the optimizer produces a tree of objects representing a query *execution plan*. There is one execution plan for each transaction.

The execution plan includes the method for getting an interface which implements positioning, retrieving, updating, and inserting an object. The next phase, the “Execution” phase, calls this method and instantiates these interfaces to actually produce the specific code which will generate the appropriate result for the TSQL statement(s).

4.4.2.2 Execution Engine

Query execution is performed by the Execution Engine and is the process of executing the plan chosen during query optimization. The Execution Engine always retrieves the execution plan from the procedure cache.

When, during the execution of a query, an attempt is made to access an object, the Execution Engine verifies the permissions are still valid. The Execution Engine creates a hash pointer into the Security Cache from the object identifier and checks the protection timestamp. If the stamp is different than the time-stamp in `syspermissions`, then the Execution Engine updates the timestamp of the protection stamp in the `DBTABLE` structure in the access methods and calls for a recompile. A recompile causes the request to begin at the top of the compile cycle.

4.4.2.3 Procedure Cache

SQL Server manages a pool of memory to store both execution plans and data buffers. The percentage of the pool allocated to either execution plans or data buffers fluctuates dynamically. The part of the memory pool used to store execution plans is called the procedure cache. The procedure cache stores previously compiled execution plans

Execution plans have two main components:

1. Query plan

The bulk of the execution plan is a reentrant, read-only data structure that can be used by any number of users. No user context is stored in the query plan. There are never more than one or two copies of the query plan in memory; one copy for all serial executions and another for all parallel executions.

2. Execution context
   Each connection currently executing a query has a data structure that holds the data specific to the execution, such as parameter values and connection information. The execution context data structures are reused. If a user executes a query and no execution context structure for that plan is in use, it is reinitialized with the context for the new user.

Certain changes in a database can cause an execution plan to be either inefficient or no longer valid, given the new state of the database. Once an execution plan is marked invalid, the execution engine will not attempt to execute the plan. A new plan must then be recompiled for the next connection that executes the query.

After an execution plan is generated, it stays in the procedure cache. SQL Server ages old, unused plans out of the cache only when space is needed. Each query plan and execution context has an associated cost factor that indicates how expensive the structure is to compile. These data structures also have an age field. Each time the object is referenced by a connection, the age field is incremented by the compilation cost factor. The lazywriter process periodically scans the list of objects in the procedure cache. On each scan the lazywriter decrements the age field for each object by one. The lazywriter process deallocates an object if three conditions are met:

- The memory manager needs memory and all available memory is currently used;
- The age field for the object is zero; and
- The object is not currently being referenced by a connection.

Because the age field is incremented each time an object is referenced, frequently referenced objects are not aged from the cache. Objects that are infrequently referenced are soon eligible for deallocation, but are not actually deallocated unless memory is needed for other objects.

4.4.2.4 Cursors

The set of rows returned by a SELECT statement consists of all the rows that satisfy the conditions in the WHERE clause of the statement. This complete set of rows returned by the statement cannot always be handled effectively by an interactive application; therefore these applications use the cursor mechanism to work with one row or a small block of rows at a time.

SQL Server supports three types of cursors:

Static Cursors -
Both cursor size and content are fixed once the cursor is populated. Creates a separate read-only copy (Snapshot Table) of the query results with the same format as the original query.

Keyset cursors -
Both cursor size and content are fixed once the cursor is populated but value changes made by the cursor owner and committed changes made by other users are visible. Creates a temporary table (Keyset Table) within the address space of the requestor that contains bookmarks, keys and timestamps. Rows deleted from the keyset are still members of the cursor; however they are marked NULL.

Dynamic Cursors -
Cursor can be opened independently from an underlying table. A fetch buffer is established for update and refresh operations. A dynamic cursor can be updated, and the actual fetch is initialized when the cursor is opened and closed when the cursor is closed.

The following operations can be performed on cursors: Declare, Open, Fetch, Update, Insert, Delete, Close, and Deallocate. Declare and deallocate perform no operations with cursor data, they establish and discard a cursor. The close operation truncates worktables from the underlying query. Open causes a query to be populated with the underlying table data for Static and Keyset cursors. For Dynamic cursors an open initializes the query but no data is transferred to the cursor. The table or fetch buffer is returned to the client as is the response to any query. The update and insert commands are used to update values and rows in the Keyset and Dynamic cursors.

4.5 SQL Server Agent

SQL Server Agent is a separate executable program that executes jobs and alerts defined by SQL Server users. It runs as a service named SQLServerAgent using the same account credentials (i.e., name and password) as the SQL Server.

A job defines a task. Each job has one or more steps; each step specifies a TSQL statement, Windows command, executable program, or replication agent. Jobs can be run once, scheduled to run at periodic intervals, or specified to run when the server is idle. Job definitions are stored in the `sysjobs` system table. The job owner is the individual who creates a job or for whom the job is created.

The Agent communicates with the SQL Server using the Windows NT Local Procedure Call (LPC) mechanism. The LPC facility is a Windows NT Executive subsystem that implements interprocess communication between two threads on the same system. The port object is the main data structure supporting the LPC mechanism. LPC communication is established and protected as described below.

The SQL Server process creates a connection port by providing a name and a security descriptor to the LPC create function. A connection port is a named port that is a server connection request point; clients can connect to the server by connecting to this port. Connection ports have names, making them visible to all Windows NT processes. These names are used as parameters in the calls to LPC made by processes to set up a communication channel with the SQL Server. The connection port name is placed in a directory specified by the attributes when the port object was created. The only access associated with a port is Port_Connect access. The Agent makes a Port_Connect request to the SQL Server's communication port to establish a communications path with the SQL Server.

As a result of the SQL Server accepting the connection request, unnamed communication ports are created by LPC. A communication port is an unnamed port used by a particular client thread to communicate with a particular server. LPC creates a client communication port and a server communication port for each connection request accepted. The Agent communication port is initialized with a flag set indicating it is a client communication port and pointers to the connection port and the SQL Server communication port. The SQL Server communication port is initialized with a flag set indicating it is a server communication port and pointers to the connection port and the Agent communication port. Communication ports are associated with a queue to which the Agent or SQL Server queues its outgoing messages. Windows NT does not provide for the sharing of communication ports.

After the Agent has established an LPC connection with the SQL Server, the Agent polls the SQL Server by logging in and requesting any entries from the `sysjobs` system table. If any jobs have been submitted, the Agent processes those jobs and logs into the SQL Server at the scheduled time to run the job. When running a job, the Agent performs the job steps as the job owner. Only the administrator can submit jobs that contain Windows NT commands. Untrusted users do not have the privilege to start an operating system command. All audit records reflect the job owner in the audit trail.

4.6 Backup and Restore

The following sections discuss database backup and restore as they relate to the database administrator and other authorized users.

4.6.1 Backup/Restore User Databases

The sysadmin is the owner of system data and is the only user authorized to perform system database backups and system database restores. The sysadmin, owner of the data, a fixed backup-database role, and other authorized users may backup and restore user databases.

The file on which a backup is stored is an ordinary Windows NT file. The file owner will be the account under which the MSSQLServer service runs. The Trusted Facility Manual (TFM) states that backup files are to be made accessible to only administrative accounts or groups.

There exists a unique situation where the ownership of a database and the tables therein, as defined in the DAC permissions list, may belong to one user prior to a backup but changes at the completion of a restore process. This happens when user “A” owns a database and its tables, and grants permission to user “B” to backup/restore the database. If user “B” has CREATE authorization, backs up the database and then restores it to a new database, user “B” becomes the owner of the database but the ownership of the tables remains with user “A”. If the WITH DBO_ONLY option was specified on the RESTORE DATABASE statement, user “A”, the owner of the tables, is not authorized to access the tables in the restored database until user “A” is granted DAC permission to the database.

The TFM and Security Features User’s Guide (SFUG) identify this situation and describe what must be done and by whom to bring ownership of the database back to the original owner.

4.6.2 Bulk Copy Program (BCP)

The following subsection describes the use of the Bulk Copy (BCP) utility program when loading data from a data file into a database or copying data from a database tables to data files.

4.6.2.1 Bulk Load

The bulk copy program, called BCP, allows users to load or unload large quantities of data from databases. The relational engine of SQL Server assures DAC compliance for the BCP utility to support loading or importing large batches of user data located on a user data file into a database table. Each batch of data processed is considered a completed transaction (see Section 4.3.3 Transaction Management on page 27). This causes all data or log pages that have been modified after they were read into the buffer cache, but whose modifications have not yet been written to disk, to be guaranteed to be written to disk. SQL Server always generates automatic checkpoints, regardless of the setting of relevant startup database options.

4.6.2.2 Bulk Unload

The storage engine of SQL Server assures DAC compliance for the BCP utility to be invoked to copy a database table to a data file in a user-specified format. Any user may invoke BCP, but access to the database table is controlled by DAC permissions.

4.6.3 Backup Devices

SQL Server backups are stored using Microsoft Tape Format (MTF) and may be stored on disk or tape. MTF enables SQL Server backups to coexist on the same media as non-SQL Server backups, provided that the backups use MTF. For example, both SQL Server backups and Microsoft Windows NT backups can exist on the same media.

SQL Server supports backup striping. A striped backup is one directed to more than a single device on a single media type. That is, a backup can be written to two tape devices but cannot be written half to a tape and the other half to a disk.

All media used for backup begins with a media header describing the media. The media header or name is usually written once and remains intact for the life of the media. This allows the data on the media to be tracked. Consistent use of media names results in correctly identifying the media and preventing errors, such as reading or over-writing media that is encrypted or protected with a password. When SQL Server overwrites the media, existing contents of the media are completely overwritten with the new backup.

4.7 Startup and Shutdown

An administrator using the Microsoft InstallShield program installs the SQL Server. To install the SQL Server, there must be a Windows NT account created under which the SQL Server runs and that account must have the four privileges described in Section 3.3.1 Operating System Privileges on page 17. During startup the first step for the SQL Server is to establish communications with the Windows NT Service Controller. Communication with the Service Controller is accomplished via LPC. After connecting with the Service Controller, the SQL Server reads in its parameters from the command line or from the Registry if none are supplied via the command line. Example parameters are default language, fiber or thread mode, login timeout, and maximum number of user connections.

Within the SQL Server process, the first action taken is to read the master database information and load the boot information. Next, the UMS is sent the fiber-mode configuration parameter identifying whether SQL Server will perform task management using fibers or threads. The first manager to start is the buffer manager, followed by the memory manager. With the memory manager started, the SQL Server allocates its memory structures and procedure cache. Next the `tempdb`, `masterdb`, and `modeldb` are all read into memory. Finally, ODS is started and SQL Server is ready to receive user connections.

During a shutdown, the SQL Server issues a checkpoint for each database and tries to obtain an exclusive lock. If an exclusive lock is obtained, then no users are using the database and it can be shutdown safely. If an exclusive lock cannot be obtained, then the database needs to be recovered during the next boot of the SQL Server. As for the SQL Server process, the shutdown process is straightforward after the databases have been closed. The SQL Server shuts down its engines and terminates.

4.8 Administrator Tools

The Administrator tools listed in this section are the applications to be used by an Administrator to manage the TCB. These are the only tools the administrator may use in the evaluated configuration. All of the tools run in user mode on the client. The Administrator tools are only in the TCB because a privileged Administrator must use them to manage the TCB.

The set of administrator tools included in the evaluated configuration is: Enterprise Manager, Query Analyzer, Client Network Utility, and Profiler. The audit analysis tool is SQL Server itself and is not further described in this section.

4.8.1 Enterprise Manager

SQL Server Enterprise Manager is a graphical tool that allows for system-wide configuration and management of SQL Server and SQL Server objects. SQL Server Enterprise Manager is used to:

- Manage logins, permissions, and users;
- Manage devices and databases;
- Back up databases and transaction logs; and
- Manage tables, views, stored procedures, triggers, indexes, rules, defaults, and user-defined data types.

SQL Server Enterprise Manager also provides:

- A scheduling engine;
- Administrator alert capability; and
- A built-in replication management interface.

4.8.2 Query Analyzer

SQL Server Query Analyzer is an interactive, graphical tool that enables an administrator to write queries, execute multiple queries simultaneously, view results, analyze the query plan, and receive assistance to improve the query performance. Additionally, the Query Analyzer allows the administrator to retrieve Transact-SQL syntax help.

4.8.3 Client Network Utility

The SQL Server Client Network Utility is used for managing the client configuration for Net-Libraries. It is used to add or change Net-Libraries in the client. The administrator uses this tool to configure the three

Net-Libraries in the evaluated configuration during installation. Configuring the client Net-Library with the Client Network Utility only affects connections to SQL Server. It does not reconfigure the network protocol used by the client operating system.

4.8.4 Profiler

The Profiler is a graphical tool that permits the administrator to view the audit trail. The administrator may use this tool or may choose to load the audit trail into a table and use the query analyzer for viewing audit data.

5 Security Architecture

This chapter describes the security architecture of SQL Server. This chapter assumes the reader is familiar with the information provided in Chapters 2-4. The first portion of this chapter describes all the protected resources in SQL Server; specifically subjects and objects. The second portion describes the SQL Server security policies: identification and authentication, roles, discretionary access control, auditing, and object reuse.

5.1 Protected Resources

This section discusses the subjects and objects in the SQL Server. The remainder of this chapter describes the security policies applied to the protected resources introduced here.

5.1.1 Subjects

Subjects within the SQL Server are either threads or fibers depending upon the execution mode selected by the administrator. A thread or fiber is initialized on behalf of a user connection to the SQL Server. Associated with each thread or fiber is a data structure which holds all security relevant information about a subject. Included in a subject's associated data structure are the user's database ID, Windows NT SID, roles, Windows NT groups, security cache, current database, and execution context.

5.1.2 Objects

An object is an entity in which untrusted processes can store user data and control sharing with other users. The set of objects in the SQL Server TCB is distinct from the objects in the Windows NT TCB. The SQL Server objects are databases, tables, columns, view definitions, stored procedures, defaults, rules, and user-defined functions. The remainder of this section identifies the common characteristics among the objects and briefly describes each object.

All objects except columns have an owner that is originally assigned to be the object's creator. The owner of the table also owns the columns in the table. Only the sysadmin or db_owner can change ownership of an object. By default, all objects are accessible only to the owner. The owner must grant access before anyone else can access an object.

A database is a collection of related information organized into tables. A table is a matrix of information organized into rows and columns. Access can be granted to specific columns in a table. Tables have a set of resources that are protected by the access to the table. These resources are: indexes, keys, constraints, and triggers. None of these resources can exist outside the table and permission cannot be granted access to these resources.

A view definition is a virtual table based upon other tables or views. The data accessible through a view is not stored in the database as a distinct object. Instead, the view definition is stored in the database. The view definition forms the virtual table returned by the view. A stored procedure is a group of TSQL statements compiled into a single execution plan.

A default, when bound to a column, specifies a value to be inserted into the column to which the object is bound when no value is explicitly supplied during an insert. A rule, when bound to a column, specifies the acceptable values that can be inserted into that column. A user-defined function provides a method for users to extend the TSQL language. A user-defined function may return a scalar or a table.

5.2 Identification and Authentication

5.2.1 SQL Server Identification

SQL Server uses several different user identifiers. These are summarized Table 5-1 SQL Server Identifiers.

USER IDENTIFIER	ALSO KNOWN AS	MNEMONIC	DESCRIPTION	SQL SERVER USAGE
Login ID	• Windows NT Username	loginname	NT user identifier--actual name of the NT login, which may be different from a database login name used by SQL Server.	• server authentication
Security Identifier	• NT Security ID (SID)	SID	NT Security Identifier that represents the Login ID	• server authentication
Server User ID	• User ID • Login • User Account	suid	SQL Server user identifier	• server authentication
SQL Server Username	• User ID • Database User ID • SQL Server login • SQL Server account	loginname	Database login	• database access • object access • auditing

Table 5-1 SQL Server Identifiers

Figure 5-1 SQL Server Identification illustrates the user identifiers used by SQL Server.

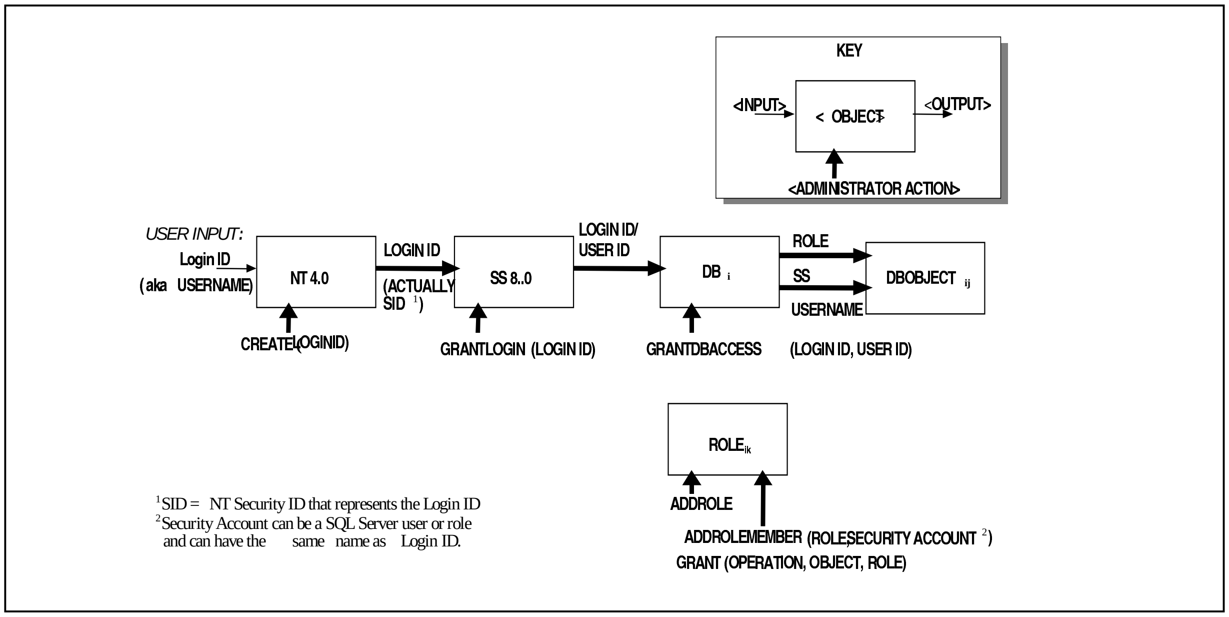

Figure 5-1 SQL Server Identification

5.2.2 Authentication to SQL Server

Two authentication modes are available for SQL Server: Windows NT authentication mode and mixed mode. In mixed mode, either Windows NT or SQL Server can perform authentication. Since only Windows NT authentication provides secure validation and encryption of passwords, auditing, password expiration, minimum password length, and account lockout after multiple invalid login requests, the evaluated configuration is limited to Windows NT authentication. With Windows NT authentication, all users, including the System Administrator, use Windows NT authentication instead of SQL Server authentication.

Restriction of SQL Server authentication to Windows NT-only authentication is done on a system-wide basis via the SecurityMode property on the Integrated Security object. The Windows NT Username for an account takes the form <`domain\name`>, where `domain` is the Windows NT domain. The SID associated with a Windows NT Username is used to represent the user in SQL Server. The SID is also mapped to a SQL Server Username in each database defined in SQL Server, and is included in database audit records.

To establish a session with SQL Server, the user must have already established a client process with Windows NT. With Windows NT authentication, the user does not have to specify another login ID or password to connect to SQL Server, because SQL Server is able to determine that the user's Windows NT authentication is valid. However, SQL Server does use an internal local User ID for access control in a database. This User ID is mapped by the administrator to the Windows NT username.

When a Windows NT user tries to connect to SQL Server, SQL Server uses Windows NT APIs, specifically the Security Service Provider Interface (SSPI), to obtain the user's SID. Login security integration, using SSPI, operates over network protocols that support trusted connections between clients and servers.

Both the SQL Server and the client attempting to perform a login communicate with their local NTLM SSPIs using LPC. When the client does not already have a connection established, it calls the NTLM SSP to get its client's credentials. The NTLM SSP gets the credentials from its local LSA. The client side sends the RPC request, including the authentication information, to the SQL Server. The SQL Server calls its NTLM SSP via LPC to perform authentication. The server-side NTLM SSP generates a challenge it sends back to the client through the SQL Server. The client receives the challenge and calls its local NTLM SSP to generate a response. The response travels back to the SQL Server's NTLM SSP and, if the authentication succeeds, the SQL Server's LSA creates a token for the user.

SQL Server gets the user account information from the token and matches it against the Windows NT accounts defined as valid SQL Server logins. If SQL Server finds a match, it accepts the connection. The server maintains user login account information in the `syslogins` table. The `syslogins` table includes the NT username and security identifier (SID). It also tracks the account's membership in any server roles (see Section 5.3 SQL Server Roles on page 45). Identification and Authentication data is protected from unauthorized access by the SQL Server DAC mechanism on the database tables containing the data.

Once authenticated to SQL Server, to obtain access to a SQL Server database a Windows NT (and hence SQL Server) user must have a corresponding user account and permissions in each database. SQL Server Usernames and permissions are database-specific, i.e., each database has a distinct set of users with corresponding permissions. While Windows NT Usernames are unique on an NT domain-wide basis, SQL Server Usernames are unique only to a database, not across the entire server. Access control permissions can be assigned by the administrator either directly to SQL Server Usernames or, alternatively, to roles. SQL Server Usernames can be assigned to roles to facilitate administration.

If no SQL Server Username that maps to the Windows NT Username were to exist, the user process could be logged onto a database under the *guest* username. However, to avoid this anonymous login, the *guest* username is to be disabled in the evaluated product.

An *alias* is a database username shared by several SQL Server Usernames. It allows more than one person to assume the same permissions in a database very much like a role. Since auditing is still performed to the level of the individual user's SID, aliases are permitted in the evaluated product.

5.2.3 Administration of User Accounts

The responsibility for managing SQL Server user accounts is divided between the server Security Administrator and the Database Access Administrator roles. A server Security Administrator can manage server-level user accounts. A Database Access Administrator can create or remove SQL Server Usernames in a database (see Figure 5-1 SQL Server Identification). A SQL Server database user account is added to a database for a Windows NT user or group, to which permissions to perform activities in the database may be granted. The TFM specifies that database access should be granted only to individual users, not to groups of users, to ensure accountability.

5.2.4 Authentication Management

A member of the server System Administrator role must specify that only Windows NT Authentication is allowed. When this is done, a SQL Server User ID will not be valid for access to SQL Server. Login accounts are created, disabled, or deleted within Windows NT by an authorized administrator via the User Manager.

Because SQL Server receives the Windows NT user's SID only at connection time, any revocation of the user's rights (i.e., revocation of permissions in `syslogins` table) will not affect the user's access to SQL Server until the next time a connection is made.

The SQL Server System Administrator specifies all the Windows NT accounts that can connect to SQL Server. The user is identified in SQL Server by Windows NT user account.

5.2.5 Password Length, Age, and Uniqueness

When an authorized Windows NT administrator sets password length, minimum and maximum age, and password uniqueness, the Windows NT OS enforces these specifications.

5.3 SQL Server Roles

This section discusses the roles used within SQL Server, both fixed (builtin) and user. The permissions granted to the fixed roles cannot be changed or dropped. The fixed roles are also privileged, while user roles are unprivileged.

5.3.1 Privileged Roles

Privileged roles are assigned to Login IDs (NT Usernames) within SQL Server and provide access to privileged operations. The set of privileged SQL Server roles includes two types of roles for system and security administration, i.e., server roles and fixed database roles. Privileged roles are described in Table 5-2 Server Role Descriptions and Table 5-3 Database Role Descriptions. These roles are privileged only with respect to SQL Server, not with respect to Windows NT. The server roles are as follows:

- `sysadmin;`
- `serveradmin;`
- `setupadmin;`
- `securityadmin;`
- `processadmin;`
- `dbcreator`; and
- `diskadmin.`

Note: It is not possible to create new fixed server roles. Roles can be created only at the database level.

The fixed database roles define permission sets within a given database. These are as follows:

- `db_owner;`
- `db_accessadmin;`
- `db_securityadmin;`
- `db_ddladmin;`
- `db_backupoperator;`
- `db_datareader;`
- `db_datawriter;`
- `db_denydatareader;` and
- `db_denydatawriter.`

5.3.1.1 Server Role Descriptions

Server Role	Description
System Administrator (`sysadmin`)	The `sysadmin` role is privileged with respect to SQL Server DAC, i.e., the role can bypass DAC. The System Administrator has numerous responsibilities related to the installation and setup of the server and management of several security functions. System Administrator tasks include any activity in SQL Server.
Server Administrator (`serveradmin`)	The `serveradmin` role can set serverwide configuration options and shut down the server.
Setup Administrator (`setupadmin`)	The `setupadmin` role can manage linked servers and startup procedures.
Security Administrator (`securityadmin`)	The `securityadmin` role can manage logins and CREATE DATABASE permissions, and read error logs.
Process Administrator (`processadmin`)	The `processadmin` role can control executing processes running in SQL Server. Only the `processadmin` role may execute the TSQL KILL statement, and this statement is the only TSQL statement that `processadmin` may execute.
Database Creator (`dbcreator`)	The `dbcreator` role can create, alter, and restore databases.
Disk Administrator (`diskadmin`)	The `diskadmin` role can manage disk files.

Table 5-2 Server Role Descriptions

5.3.1.2 Database Role Descriptions

Database Role	Description
Database Owner (`db_owner`)	The `db_owner` role has all permissions in the database, i.e., the permissions of db_owner span those of all the other fixed database roles.
Database Access Administrator (`db_accessadmin`)	The `db_accessadmin` role can add or remove user IDs.
Database Security Administrator (`db_securityadmin`)	The `db_securityadmin` role can manage all permissions, object ownerships, roles and role memberships.
Database DDL Administrator (`db_ddladmin`)	The `db_ddladmin` role can issue ALL DDL, but cannot issue GRANT, REVOKE, or DENY statements.
Database Backup Operator (`db_backupoperator`)	The `db_backupoperator` role can issue DBCC, CHECKPOINT, and BACKUP statements.
Database Data Reader (`db_datareader`)	The `db_datareader` role can select all data from any user table in the database.

Database Role	Description
Database Data Writer (`db_datawriter`)	The `db_datawriter` role can modify any data in any user table in the database.
Database Deny Data Reader (`db_denydatareader`)	The `db_denydatareader` role can deny or revoke SELECT permissions on any object.
Database Deny Data Writer (`db_denydatawriter`)	The `db_denydatawriter` role can deny or revoke INSERT, UPDATE, and DELETE permissions on any object.

Table 5-3 Database Role Descriptions

5.3.2 User Roles

User roles are those created in an operational database. They are similar in function to groups, in that they combine users having the same access permissions. User roles specify the types of action an ordinary user may perform on a database object, and are attached to database objects.

A role is assigned to a set of users who are to be granted a permission set in a database. Every user in a database belongs to the `public` database role. In addition, a Database Security Administrator can create user roles. Roles are granted to and revoked from individual login accounts, and are stored with other user account information in the system tables.

When a user logs in, any user roles granted to that user are activated. The user cannot turn on or off the roles of which the user is a member.

5.4 Discretionary Access Control

SQL Server implements a DAC policy on its objects separate from the Windows NT DAC policy. Access can be granted to DBMS objects to the granularity of users, groups or roles. SQL Server data is protected to the individual column level.

There are two system tables that are used to store DAC information; both of these system tables exist for all databases. SQL Server stores the DAC permission to use a database in the `sysusers` system table. Within each database, SQL Server stores its DAC-related information for the objects within the database in the `syspermissions` system table. Each row in the `syspermissions` system table can be thought of as an ACL as it has a list of users, groups, and roles and their associated permissions for an object. Whenever a user attempts to access a DBMS object, that user's security cache is checked and if necessary, the associated row in the `syspermissions` table is checked to ensure the user has access. The objects in the DBMS which have DAC applied are databases, tables, columns, views, stored procedures, rules, and defaults. Indexes and triggers are properties of tables and are protected by the DAC on their associated tables.

There are two ways a user can gain access to an object. The first is to be granted access to the object and the second is to have access based upon a database role. The remainder of this section discusses the specific types of DAC permissions and how each permission fits into the overall DAC policy.

5.4.1 Permissions

There are three types of permissions that can be granted to an object – object, statement, and implied. Object permissions are those permissions assigned to users based upon access to a particular object. DBMS objects with object permissions are databases, tables, columns, views, and stored procedures. Table 5-4 Object Permissions shows the types of object permissions available on each object.

Permission	Object Type
USE	Database
SELECT, INSERT, UPDATE, DELETE	Entire table or view
SELECT, UPDATE	Columns of table or view

Permission	Object Type
INSERT, DELETE	Entire table or view
EXECUTE	Stored procedure

Table 5-4 Object Permissions

The USE permission permits a user to access a database. The SELECT permission allows a user to read the data in a table, column, or view depending upon where the permission was granted. The INSERT permission allows a user to add a row to a table or view, while DELETE removes rows. The UPDATE permission is used to changes data already in a table or column. EXECUTE permission allows a user to invoke a stored procedure.

Statement permissions are those that allow a user to issue a particular statement. Statement permissions are applied to the SQL statement itself rather than an object defined in a database. Like object permissions, statement permissions are stored in the `syspermissions` system table. The statement permissions are:

- CREATE DATABASE;
- CREATE TABLE;
- CREATE VIEW;
- CREATE PROCEDURE;
- CREATE DEFAULT;
- CREATE RULE;
- CREATE FUNCTION;
- BACKUP DATABASE; and
- BACKUP LOG.

There is no specific restore permission in the SQL Server. To restore a database over an existing database, the user must have BACKUP DATBASE permission. To restore without overwriting a database, the user must have CREATE DATABASE permission.

Implied permissions are given to users based upon predefined DBMS roles or owners of database objects. There are three pre-defined roles that have implied permissions – sysadmin, db_owner, and object owner. The sysadmin role has permissions to do or see anything within the SQL Server. The db_owner role has permissions to control all data within a database. For example, the db_owner controls who can create tables and what users can use the db_owner's database. The third category of implied permissions is for object owners. Owners of objects have a full set of permissions on the objects they create and can grant permissions to other users to use their objects. By default, the creator of an object is assigned to be its owner. The sysadmin or db_owner can choose to reassign ownership to another user or group. Specific rule and default permissions are implied permissions. Rule owners have permission to bind and unbind a rule from a column or user-defined data type. Default owners also have permission to bind and unbind a default from a column or user-defined data type. Binding a rule or default applies the applicable value to the column or user-defined data type. Rule and default owners also have permission to drop their associated rules or defaults. The owner of a user-defined function also has several implied permissions. The owner may alter or delete the user-defined function.

There is one set of TSQL commands that is handled differently with respect to the DAC policy. These are DBCC commands. The DBCC commands act as the database consistency checker SQL Server. These statements check the physical and logical consistency of a database. Many DBCC statements can fix detected problems. Most of these commands are restricted to the administrator. All access checks for DBCC commands are imbedded within the commands and cannot be changed.

5.4.2 Granting and Revoking Permissions

Object owners assign permission using a grant and revoke mechanism. Object owners give access to other users by granting those users a specific permission. Conversely, object owners can choose to deny a specific access to a user as well. In each of these cases, an entry is added to the `syspermissions` system table with the appropriate permission granted or denied. When granting a permission to another user, the grantor can choose to give the permission with the *grant option*. The grant option allows the recipient to grant that same permission to other users.

To remove an entry from the `syspermissions` table, users issue the revoke command. The revoke command results in the associated granted or denied permission being deleted. The revoke command is issued by a user that granted a permission. In the case where a permission has been granted with the grant option, the revoke *cascades*. Permission revoked from a user that has used the grant option causes the original permission to be removed, as well as all permission granted by that user. For example, assume User A granted User B select on a table 1 with the grant option. User B then grants User C and User D the select option on table 1. When User A revokes User B's access, User C and User D also lose access. Permission can be revoked on an object when a user has the object open. The revocation will not have an affect until the next time the object is accessed.

5.4.3 Object Dependencies

There is a dependency between database and table level permissions. Before a user can access a table within a database, the user has to be given permission to the database. Similarly, having accesses to a database does not give a user access to any of its tables; users must be explicitly given permission to access specific tables.

Views depend on other tables or views. Stored procedures depend on tables, views, or other stored procedures. These two types of dependencies are called *ownership chains*. Often the same user owns all views, stored procedures, and the objects they depend upon. In this case, the SQL Server does not make the DAC check on any of the dependent objects; only access to the view or stored procedure is checked.

In the case where another user owns a dependent object, the ownership chain is broken and SQL Server must make additional DAC checks. If the ownership chain of a stored procedure or view is broken, SQL Server checks permissions on each object in the chain whose next lower link is owned by a different user. In this way, SQL Server allows the owner of the original data to retain control over who is authorized to access it. Figure 5-2 Ownership Chain Example shows an example of an ownership chain. Jane creates a procedure and gives Sally access. In this example, SQL Server will perform access checks on procedure1, procedure2, and table2.

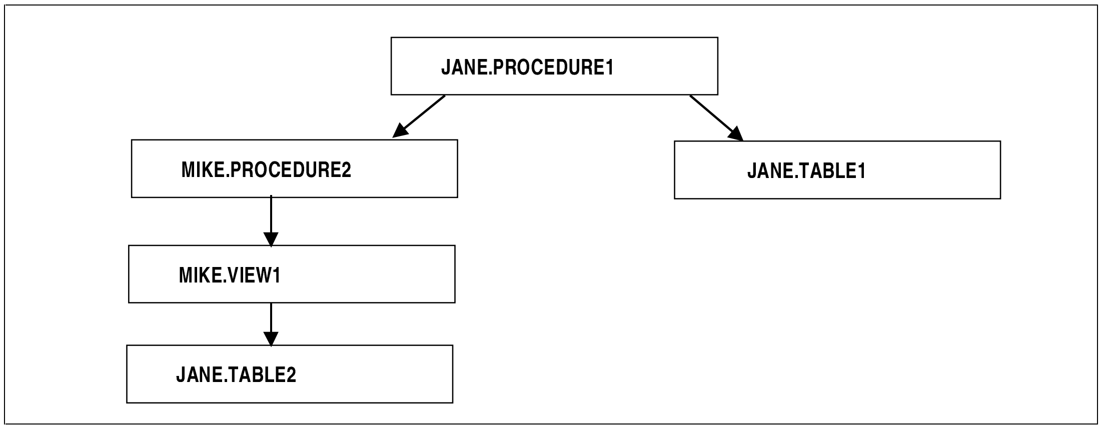

Figure 5-2 Ownership Chain Example

5.4.4 DAC Algorithm

Before gaining access to an object, a user must pass a series of DAC checks. A few general rules apply. The first rule is that a denied permission always takes precedence. The denial can be at a user or group level and will override any granted permission. A revoked permission removes only the granted or denied permission at the level revoked (user or group). The same permission granted (or denied) at another level such as a group or role containing the user, group, or role still applies. A granted permission removes the denied or revoked permission at the level granted. The same permission denied at another level, such as a group containing the user, still applies. The DAC algorithm is as follows:

- For the set of IDs associated with a user, read though all entries in the `syspermissions` table looking for specific access for:
  - sysadmin
  - db_owner
  - User ID
  - ID of any Windows NT groups
  - ID of any SQL Server roles
- In any instance if a matching entry is found that denies access, access denied is returned.
- If the requested access is found in the `syspermissions` table and no deny access is found, grant the access.

5.5 Audit

SQL Server provides the ability to audit all security relevant events that occur within the DBMS. There is a configuration option that enables C2 auditing available to the administrator. This section discusses how audit records are created, what events are audited, and the information contained in the audit records.

5.5.1 Audit Record Generation

Audit is turned on or off by setting a C2 trace flag using the sp_configure system stored procedure. If the flag is set to on, then all C2 relevant events are audited. Similarly, if the flag is off, no events are audited. The C2 flag is read by the SQL Server at startup. If auditing is turned on, the SQL Server allocates a memory structure called a Ctrace to manage the collection of the audit records and the output of those records to disk. Within the Ctrace is a list of events to audit, columns of data to collect, and a filename of where to write the collected events. SQL Server also maintains two global audit buffers to store audit records until the data can be flushed to disk.

Audit producers in the ODS event handlers and relational engine generate audit events. The producers in these two areas check the Ctrace structure to determine if their particular action requires an audit event be generated. If the audit switch is on for the producer's event, the producer generates a record and adds it to a field in the associated subject's Execution Context.

Auditing occurs at compilation time. The timestamp in the audit records reflects the time at compilation, not execution. If during the execution checks, a change in the protection timestamp has occurred, a re-compilation is necessary and new audit records will be generated for the new access checks.

Memory is flushed from the audit buffers on a periodic basis. The records are written to a file configured at audit startup. Windows NT DAC protects the file and only permits the SQL Server and Administrators access.

5.5.2 Audited Events

With the C2 audit flag set, the SQL Server audits all security relevant events including logins, object accesses, and administrator actions. All audit records are rows in a SQL Server table with each row having the following information:

- Type of event being recorded;
- Success or failure;
- SQL Server name;
- Date and Time;
- User application name;

- Windows NT user name and SID;
- SQL Server PID; and
- Client computer name that originated request.

There are 15 categories of audit events. Table 5-5 Audit Categories enumerates the different types of auditable events and identifies any information in the associated audit record that was not identified as common information.

AUDIT CATEGORY	AUDIT RECORD CONTENTS
Login/logout	<nothing additional>
Grant/revoke/deny for object permission	Database ID, database name, database user name, object name, object owner, object permissions, statement text
Grant/revoke/deny for statement permission	Database ID, database name, database user name, statement type, statement text
Grant/revoke/deny a Windows NT account login	Target login SID, target login name
Modify a login property (default language or default database)	Target login SID, target login name
Add/remove a login to a fixed SQL Server role	Target login SID, target login name, target role name
Add/remove a database user	Database ID, database name, database user name, Target login SID, target login name, target database user name, target role name
Add/remove a member to a database role	Database ID, database name, database user name, target database user name, target database role name.
Add/drop a database role	Database ID, database name, database user name, target database role name
Statement permission used	Database ID, database name, database user name, statement type, and statement text
Object permission used	Database ID, database name, database user name, object name, object owner, object permissions, statement text
Backup/restore	Database ID, database name, database user name, statement text
DBCC command used	Database name, database user name, statement text
Audit flag modifications	Statement text
SQL Server start/stop/pause	<nothing additional>

Table 5-5 Audit Categories

5.5.3 Audit Selection

The audit trail is stored on disk in binary format. In order to read the audit trail, the administrator must run either the Profiler graphical tools or use a system-stored procedure to convert the audit trail from binary format into a SQL Server table. Once the trail is converted into a table, the administrator can run SQL queries against the audit data. The administrator can select audit data based on any field in the database including user name.

5.5.4 Audit Data Loss

The audit configuration for C2 provides an automatic audit log rollover capability. When a file containing an audit fills, the SQL Server automatically creates a new file named audit_YYYYMMDDHHSS_sequence number>. The SQL Server will continue to rollover files until there is no more disk space available to it. Once the SQL Server cannot create new audit files, it stops. The administrator must then free some disk space and restart the SQL Server.

While processing, SQL Server stores its audit records in Execution Context structures. If SQL Server crashes during processing, the maximum number of audit records that could be lost is 128K worth of data.

5.6 Object Reuse

Object reuse concerns the allocation of resources that have been used to store information and then released back to the system for future use. A subject must not be presented data for which the subject has no access through an interface available to untrusted user applications, even if the interface is used in an unusual or non-typical way or through undocumented, yet publicly available, interface options.

Fundamentally, the object reuse security policy comprises four elements:

1. All user data on disk is managed by an evaluated Windows NT;
2. All user data within the address space of SQL Server is managed by the buffer manager which overwrites user data buffers upon allocation;
3. When user data is transferred between memory and disk, it is transferred in fixed-size pages; and
4. No interface is provided to access user data outside of constraints specified in the PFS and IAM tables.

5.6.1 Databases

A database is composed of at least two files managed by the Windows NT. No two databases share a single file, so that each file is unique to a database. Whenever a database is created, Windows NT clears all residual information in the underlying files.

5.6.2 Pages

Pages are fixed-length pieces of an underlying file. When a page is retrieved from disk, it is retrieved as a full page and placed into a fixed-length in-memory buffer. Before a page is allocated, an in-memory structure (page buffer), that is a full page in size, is created for the page information. This in-memory structure is filled with zeros upon allocation. Also a full page is always read into memory and written from memory to disk.

5.6.3 Rows

Rows cannot span pages; therefore every action on a row must be encapsulated within an allocated page. When a row is added to a table, the row will either be added in an already assigned page or an extent will be created. If the row causes the creation of an extent, page and buffer management insure the 8 KB areas of pages or page buffers are clear. If the row is added to an existing page, there is no interface provided to read outside of the row. When a row is deleted, it is removed from the Index Allocation Map (IAM) and the PFS is updated by placing the page area that was assigned to the row back on the free list. There is no interface provided to allow client applications to access a page area of a deleted row, or a page area outside the row limitations.

5.6.4 System Data Structures

The in-memory data structures used by the Access Methods manage database objects in memory. These in-memory structures such as the `DBTABLE`, `DES`, and `SDES` contain user data. However, there is no exported interface to access or manipulate these structures.

5.6.5 TDS Packets

TDS Packets are the unit of data transferred over one of the two supported Windows NT Interprocess Control mechanisms (IPC); named pipes or sockets. Since the services provided by the evaluated version of Windows NT provides object reuse protection for both IPC mechanisms, the only potential issue concerning object reuse (when sending and receiving TDS packets) is whether residual data within a previous TDS packet buffer can be sent over a connection with the current TDS packet.

For every valid connection, also called a session that occurs after a successful SQL Server login has been completed, ODS calls UMS to create a separate thread for the connection with its own per session state and its own execution context. All TDS packets are communicated through TDS buffers. All TDS buffers communicated over a specific connection are stored within the address space of the specific connection and are isolated from all other connections, therefore there is no way for a user, communicating over one connection, to receive residual information from a buffer previously communicated over another connection.

6 Assurances

Assurance that the security mechanisms work correctly to enforce the system defined security policy is a fundamental requirement of a trusted product. The following sections describe the assurance features of SQL Server. A description of the mechanisms, which ensure that the SQL Server TCB cannot be corrupted, is provided, followed by a description of the supporting evidence (i.e., design documentation and specifications) for those mechanisms. The security testing performed by Microsoft is also described.

6.1 System Architecture

This section discusses the mechanisms used by SQL Server to protect itself from unauthorized modification.

The composite TCB is composed of the Windows NT TCB and the SQL Server TCB. SQL Server runs as a trusted subject with respect to the operating system. SQL Server only uses the following Windows NT privileges:

- AssignPrimaryToken;
- IncreaseQuota;
- TCB ; and
- ServiceLoginRight.

The Windows NT operating system is not compromised by SQL Server's use of these privileges. They are primarily used for running the SQL Server as a service, and logon for SQL Server Agent. SQL Server's use of these privileges is further described in Section 3.3.1 Operating System Privileges on page 17.

The Windows NT TCB implements a domain for its own execution that protects itself from tampering. The SQL Server TCB relies on the Windows NT TCB to provide process isolation and protect it from external tampering. Specifically, the SQL Server TCB runs as a protected server on the Windows NT operating system and uses the operating system's process isolation mechanisms to maintain and protect its execution domain. TCB protected servers run in a user-mode TCB process. Each protected server provides a set of functions available to untrusted clients. Access to these functions is via a client-server model using Windows NT RPC.

SQL Server relies on the Windows NT DAC mechanism to protect its data and executable files from tampering. Only the SQL Server service account may access data files directly. All SQL Server data is stored in the MSSQL directory where SQL Server has *full control* access. No other user accounts have access to this directory. Untrusted users only have *execute* access to SQL Server executable files.

All subjects within the SQL Server have associated data structures, which contain the user context. One assocaited structure contains the SQL Server unique user name and Windows NT SID. This structure is used to provide process isolation within SQL Server.

All attempts to access SQL Server protected objects are subject to the DAC and audit requirements. See Audit and DAC sections for more information.

6.2 Security Testing

The vendor's test philosophy is one of API coverage for security relevant APIs. The vendor test suite is composed of two categories, stored procedures and TSQL commands. For each category, there are associated access control, object reuse, and system architecture tests. In addition to the main categories, the vendor has an audit test to specifically test the audit functions.

Each functional test suite is designed to provide coverage both in breadth and depth. The breadth of coverage is realized from direct testing of relevant APIs. The depth of coverage is realized by sets of test cases involving many combinations of parameters. Together, these functional test suites are designed to provide coverage of the security functions of the TCB.

Microsoft's SQL Server tests include a number of test suites. A brief summary of each is included in the table below. The test suites are mostly automated with very few exceptions. The tests are run using a Microsoft created test harness and the results may be examined using a Microsoft developed analysis tool.

TEST SUITE NAME	TEST SUITE DESCRIPTION
Stored Procedures – System Architecture	This test suite tests that stored procedures only the administrator can execute are in fact so restricted.
Stored Procedures – Access Control	This test suite tests access is enforced properly on stored procedures and the objects they internally reference.
TSQL – Access Control	This test suite tests access is granted, revoked, and enforced properly on objects manipulated in the SQL Server.
TSQL – System Architecture	This test suite tests that TSQL interfaces restricted to the administrator are in fact so restricted.
TSQL – Object Reuse	This test suite ensures that any resources allocated have been properly subjected to the reuse mechanism (cleared at allocation).
Audit	This test suite tests the audit mechanism including generation of audit events and shutdown if audit trail is full.
Undocumented APIs - System Architecture	This test suite tests that undocumented APIs restricted to the administrator are in fact so restricted.

Table 6-1 Test Suites

6.3 Design Documentation

The Philosophy of Protection for SQL Server is described in the Managing Security section of the SQL Server Books Online and the Functional Specification for Auditing in SQL Server. This philosophy addresses user authentication, database object protection, and auditing. It also depicts the security model of SQL Server wherein users must be authenticated for system level access and then are subject to the reference monitor prior to accessing any protected objects.

The translation of this philosophy to the TCB has been determined using documentation in many forms. Documentation used in evaluating this system against the TDI class C2 requirements and in mapping the philosophy of protection to the TCB are as follows:

- SQL Server Books Online – this set of documentation provides a compressive discussion of the SQL Server from an administrator and user point of view. It details how to administer and use the SQL Server, as well as provides programming reference information, including TSQL.

- Microsoft SQL Server Introduction book – this publicly available book, sponsored by Microsoft, provides insight into the design of the SQL Server product.

- Numerous design specifications, design notes, and white papers – this set of documentation includes design information for many components of the SQL Server. Some of this material is dated and is either no longer valid or has required confirmation from other sources listed below.

- Microsoft Developer Network (MSDN) Library –This documentation (on CD-ROM or Microsoft web pages) is readily searchable and provides hyper-links for easy browsing. It contains current information about public APIs and some major component design details.

- Source Code Headers – source code headers are useful primarily to confirm or to determine whether existing, but dated, information is still valid and to determine deviations from documented design that need to be addressed (e.g., by going into the code itself). In addition, source code headers are useful to confirm information derived from interaction with SQL Server developers.

- Source Code – source code was examined by the evaluation team to provide additional clarification when appropriate.

Additionally, hands-on experience with actual SQL Server systems configured per the evaluation proved invaluable in gaining confidence and in confirming the design of the system as described elsewhere. In addition, interaction with developers (essentially training) served to help expedite evaluation analysis by summarizing complicated features and providing pointers regarding where to start and specific topics to reference.

7 Evaluation as a C2 System

7.1 Discretionary Access Control

Requirement

TCSEC

The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals, or defined groups of individuals, or by both, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access. These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.

TDI

The discretionary access control requirements apply as stated in the TCSEC to every TCB subset whose policy includes discretionary access control of its subjects to its objects. Any TCB subset whose policy does not include such discretionary access control is exempt from this requirement.

Applicable Features

SQL Server uses the grant, revoke, and deny mechanism to control access between named users and named objects. The use of grant, revoke, and deny allows users to specify and control the sharing of objects with other users, groups of users, or both. The grant, revoke, and deny mechanism has the ability to specify access rights to be granted or denied at the granularity of an individual user. By default, only an object's creator is given access to a named object. Only authorized users – those with appropriate DAC permission, the owner, or those with DAC privileges can grant access. In order to limit the propagation of access rights, the revoke permission is provided. See Section 5.4 Discretionary Access Control on page 47 for more details.

Conclusion

SQL Server satisfies the C2 Discretionary Access Control requirement

7.2 Object Reuse

Requirement

TCSEC

All authorizations to the information contained within a storage object shall be revoked prior to initial assignment, allocation or reallocation to a subject from the TCB's pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject's actions is to be available to any subject that obtains access to an object that has been released back to the system.

TDI

This requirement applies as stated in the TCSEC to every TCB subset in the TCB.

Applicable Features

SQL Server ensures all resources visible at the TCB interface provide no residual user data. All resources are cleared upon allocation. Files managed by the SQL Server are cleared according to the Windows NT object reuse policy. See Section 5.6 Object Reuse on page 52 for more details.

Conclusion

SQL Server satisfies the C2 Object Reuse requirement

7.3 Identification and Authentication

Requirement

TCSEC

The TCB shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall use a protected mechanism (e.g., passwords) to authenticate the user's identity. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual.

TDI

This requirement applies as stated in the TCSEC to the entire TCB. The cooperative action of the TCB subsets making up the TCB must satisfy the requirement.

If the TCB is composed of TCB subsets, one TCB subset may either rely on a mechanism in another TCB subset to provide identification and authentication services or provide the services directly. Each TCB subset may maintain its own identification and authentication data or one central repository may be maintained. If each TCB subset has its own data, then the information for each individual user must be consistent among all subsets.

Applicable Features

SQL Server relies on Windows NT to perform authentication for it. SQL Server uses the token provided by Windows NT to map the Windows NT user to a SQL Server user. This mapping provides the ability to associate individuals with auditable actions within the SQL Server. See Section 5.2 Identification and Authentication on page 42 for more details about identification and authentication.

Conclusion

SQL Server satisfies the C2 Identification and Authentication requirement.

7.4 Audit

Requirement

TCSEC

The TCB shall be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity.

TDI

This requirement applies as stated in the TCSEC to the entire TCB. The cooperative action of the TCB subsets making up the TCB must satisfy the requirement.

A TCB subset may maintain its own security audit log, distinct from that maintained by more primitive TCB subsets, or it may use an audit interface provided by a different TCB subset allowing the audit records it generates to be processed by that TCB subset.

If the TCB subset uses different user identifications than a more primitive TCB subset, there shall be a means to associate audit records generated by different TCB subsets for the same individual with each other, either at the time they are generated or later.

Auditable events, in the case of a database management system, are the individual operations initiated by untrusted users (e.g., updates, retrievals, and inserts), not just the invocation of the database management system. The auditing mechanism shall have the capability of auditing all mediated accesses, which are visible to users. That is, each discretionary access control policy decision shall be auditable. Individual operations performed by trusted software, if totally transparent to the user, need not be auditable. If the total audit requirement is met by the use of more than one audit log, a method of correlation must be available.

Applicable Features

The SQL Server TCB creates and maintains an audit trail of identification and authentication, access to objects it protects, deletion of protected objects, administrator actions, and other security relevant events. Audit information is recorded in a file that is accessible only if appropriate DAC permission is obtained. Only administrators are allowed access to the audit trail. All audit records include the time, user, event type and success or failure, among other details. Identification and authentication audit records include the computer name originating the event. Audit records for access to and deletion of named objects include the object name. The audit mechanism allows for an administrator to post-select the audit events in the system. The administrator imports the audit trail into a SQL Server database and then uses the SQL Server to view the audit trail. This tool allows the administrator to selectively view the audit actions of users based on individual identity. The audit subsystem can be configured to prevent the loss of audit records. See Section 5.5 Audit on page 50 for more details.

Conclusion

SQL Server satisfies the C2 Audit requirement.

7.5 System Architecture

Requirement

TCSEC

The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). Resources controlled by the TCB may be a defined subset of the subjects and objects in the ADP system. The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements.

TDI

This requirement applies as stated in the TCSEC to every TCB subset, with the following additional interpretations.
The TCB must provide domains for execution that are allocated to and used by TCB subsets according to the subset-domain condition for evaluation by parts. A most primitive TCB subset must provide domains for execution. A less primitive TCB subset must make use of domains provided by a more primitive TCB subset. A less primitive TCB subset may provide further execution domains but is not required to do so.

If the TCB is composed of multiple TCB subsets, this requirement applies to each TCB subset.

Applicable Features

The composite TCB is composed of the Windows NT TCB and the SQL Server TCB. The Windows NT TCB implements a domain for its own execution that protects it from tampering. The SQL Server TCB relies on the Windows NT TCB to provide process isolation and protect it from external tampering. Specifically, the SQL Server TCB runs as a protected server on the Windows NT operating system and uses the operating system's process isolation mechanisms to maintain and protect its execution domain. SQL Server relies on the Windows NT DAC mechanism to protect its data from tampering. Only the SQL Server service account may access data files directly. All subjects within the SQL Server have data structures to separate them from other subjects. All attempts to access SQL Server protected objects are subject to the DAC and audit requirements. See Section 6.1 System Architecture on page 54 for more information.

Conclusion

SQL Server satisfies the C2 System Architecture requirement.

7.6 System Integrity

Requirement

TCSEC

Hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the TCB.

TDI

This requirement applies as stated in the TCSEC to every TCB subset that includes hardware or firmware. Any TCB subset that does not include hardware or firmware is exempt from this requirement.

Applicable Features

This requirement is handled entirely by Windows NT.

Conclusion

SQL Server satisfies the C2 System Integrity requirement.

7.7 Security Testing

Requirement

TCSEC

The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. Testing shall be done to assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanisms of the TCB. Testing shall also include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorized access to the audit or authentication data.

TDI
This requirement applies as stated in the TCSEC to the entire TCB. If a TCB consists of TCB subsets meeting the conditions for evaluation by parts, the satisfaction of the requirements by each TCB subset satisfies the requirement for the entire TCB. Otherwise, security testing of the entire TCB must be performed (even if the results of testing the individual TCB subsets were available).

Applicable Features

The evaluation team executed the vendor's test suite (automatic and manual) on the Desktop, Standard, and Enterprise versions of the SQL Server. The evaluation team analyzed the results from the test runs and worked with Microsoft to correct any anomalies found. The team also performed its own independent testing to search for obvious flaws. This team testing included a search of public domain sources for known flaws in SQL Server.

Conclusion
SQL Server satisfies the C2 Security Testing requirement.

7.8 Security Feature's Users Guide

Requirement

TCSEC
A single summary, chapter, or manual in user documentation shall describe the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another.

TDI
This requirement applies as stated in the TCSEC to every TCB subset in the TCB. This collection of guides must include descriptions of every TCB subset in the TCB and explicit cross-references to other related user's guides to other TCB subsets, as required. In addition, interactions between mechanisms within different TCB subsets must be clearly described.

Applicable Features

Microsoft has a SFUG that describes the protection mechanisms available to users. It describes how to log on, how to change a password, and how to lock and unlock the computer. It also describes how to use the DAC mechanism to protect SQL Server named objects, including explaining the access revocation policy.

The SFUG is entitled *C2 Administrator's and User's Security Guide* and is available on the Microsoft web site at http://www.microsoft.com/security.

Conclusion

SQL Server satisfies the C2 Security Features Users Guide requirement

7.9 Trusted Facility Manual

Requirement

TCSEC
A manual addressed to the ADP system administrator shall present cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given.

TDI

This requirement applies as stated in the TCSEC to the TCB and to every TCB subset in the TCB.

This requirement can be met by providing a set of manuals, one for each distinct (non-replicated) TCB subset. Each manual shall address the functions and privileges to be controlled for the associated TCB subset. Additionally, it must clearly show the interfaces to, and the interaction with, more primitive TCB subsets. The manual for each TCB subset shall identify the functions and privileges of the TCB subsets on which the associated TCB subset depends. Also, the TCB subset's manual shall identify which, if any, configuration options of the more primitive TCB subsets are to be used for the composite TCB to operate securely.

Any pre-defined roles supported (e.g., database administrator) by the TCB subset shall be addressed.

The means for correlating multiple audit logs and synonymous user identifications from multiple TCB subsets (if such exist) shall also be addressed.

The trusted facility manual shall describe the composite TCB so that the interactions among the TCB subsets can be readily determined. Other manuals may be referenced in this determination. The manuals that address the distinct modules of the TCB and the generation of the TCB need to be integrated with the other trusted facility manuals only to the extent that they are affected by the use and operation (versus the development) of the composite TCB.

Applicable Features

Microsoft has a TFM that describes the evaluated configuration and how to achieve it, privileges and their assignment, and the detailed audit record structure. In addition, the TFM also enumerates the tools available for the administrator and summarizes their use. Appropriate instructions and cautions are provided for operating the system in the evaluated configuration.

The TFM for the SQL Server is separate from the Windows NT TFM. It identifies the privileges SQL Server depends upon from Windows NT. All builtin database roles are described and the means for correlating multiple audit logs and synonymous user identifications between SQL Server and Windows NT are also addressed.

The TFM is entitled *C2 Administrator's and User's Security Guide* and is available on the Microsoft web site at http://www.microsoft.com/security.

Conclusion

SQL Server satisfies the C2 Trusted Facility Manual requirement.

7.10 Test Documentation

Requirement

TCSEC

The system developer shall provide to the evaluators a document that describes the test plan, test procedures that show how the security mechanisms were tested, and results of the security mechanisms' functional testing.

TDI

This requirement applies as stated in the TCSEC to the composite TCB.

Applicable Features

The test documentation is divided into three categories: TSQL, stored procedures, and audit. For each test category, the test documentation includes a high-level test plan describing the testing philosophy for the associated APIs. There are high-level test plans for TSQL, stored procedures, undocumented APIs, and audit.

Within each test category are specific test plans that describe the test procedure and test cases for each API. There are specific test plans for the following categories:

- TSQL – Access control, object reuse, system architecture;
- Stored Procedures - Access control, object reuse, system architecture;
- Undocumented APIs – System architecture; and
- Audit

Also in each test plan is a reference to the SQL source code file for each test case and the expected results for the test case.

Conclusion

SQL Server satisfies the C2 Test Documentation requirement.

7.11 Design Documentation

Requirement

TCSEC

Documentation shall be available that provides a description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the TCB. If the TCB is composed of distinct modules, the interfaces between these modules shall be described.

TDI

This requirement applies as stated in the TCSEC to the composite TCB.

Applicable Features

The Philosophy of Protection for SQL Server is described in the Managing Security section of the SQL Server Books Online and the Functional Specification for Auditing in SQL Server. This philosophy addresses user authentication, database object protection, and auditing. It also depicts the security model of SQL Server wherein users must be authenticated for system level access and then are subject to access checks prior to accessing any protected objects.

The translation of this philosophy to the TCB has been determined using documentation in many forms. Documentation used in evaluating this system against the TDI class C2 requirements and in mapping the philosophy of protection to the TCB includes SQL Server Books Online, Microsoft SQL Server Introduction book, numerous design specifications, design notes, white papers, Microsoft Developer Network (MSDN) Library, Source Code Headers, and Source Code. The interfaces between the SQL Server and Windows NT TCB's are documented in the design notes and design specifications. See Section 6.3 Design Documentation on page 55 for more details.

Conclusion

SQL Server satisfies the C2 Design Documentation requirement.

6 ITCSEC Mapping

This section identifies C2-relevant Interpreted Trusted Computer System Evaluation Criteria (ITCSEC) requirements and summarizes how they are addressed. The following table is organized by C2 requirement. Each section identifies the C2-relevant interpretations and summarizes how the requirements of each interpretation are addressed.

Several ITCSEC requirements are met by TFM entries. Currently the team does not have an acceptable version of the TFM. The evaluation team believes, however, that if Microsoft produces a document with the agreed content that the requirement will be satisfied.

C2 REQUIREMENT & INTERPRETATION			DISPOSITION
Audit			
	C1-CI-04-84		SQL Server audits all events that are security-relevant in the context of the class C2 requirements.
	C1-CI-07-84		This interpretation represents no class C2 requirement. Rather it affects class B2.
	C1-CI-02-85		SQL Server allows selection of audit records based on user identity (and other criteria). Security levels and configuration management are not applicable to class C2 products.
	C1-CI-02-89		SQL Server allows administrators to select whether the system will cease performing auditable actions or to simply lose audit records when the audit subsystem encounters a predictable problem (e.g., the audit log is full). In any case, the circumstances and amount of audit records that could be lost is identified in the TFM.
	I-0004	Enforcement of audit settings consistent with protection goals	All audit settings are immediately enforced for subsequent (i.e., not already initiated) events.
	I-0005	Action for audit log overflow	SQL Server can be configured to cease performing auditable actions if the audit trail fills. This option is described in the TFM.
	I-0006	Audit of user-id for invalid login	This interpretation represents no requirement.
	I-0043	Auditing use of named pipe	It is not clear what “similar to unnamed pipes in UNIX systems” means. However, SQL Server audits the introduction of objects it protects into a user’s address space – per the class C2 audit requirement.
	I-0073	OK to audit decision regardless of whether action completed	This interpretation represents no requirement.
	I-0286	Auditing unadvertised TCB interfaces	SQL Server audits all security relevant events that occur from the use of the TCB interface.
Design Documentation			
	I-0192	Interface manuals as design documentation	More than interface reference manuals document the TCB design.

I-0193	Standard system books as design documentation	This interpretation applies to documentation other than books for SQL Server. All documentation fitting into this category has been confirmed with other documents, developer interaction, source code analysis, etc.
Discretionary Access Control		
C1-CI-06-84		SQL Server does not use passwords in its DAC mechanisms.
C1-CI-02-86		The specific implementation is not applicable to SQL Server.
C1-CI-03-86		SQL Server protects (i.e., allows only access to authorized users) all named objects by default.
I-0002	Delayed revocation of DAC access	DAC changes are immediately effective for subsequent object access attempts. The SFUG describes the access revocation policy.
I-0020	DAC authority for assignment	Users authorized by DAC can make named objects sharable. There are no roles defined whose purpose is to control sharing.
I-0053	Public objects and DAC	This interpretation effectively represents no class C2 requirement.
I-0222	Passwords not acceptable for DAC	SQL Server does not use passwords in its DAC mechanisms.
I-312	Set-ID and the DAC requirement	This interpretation represents no requirement.
Identification and Authentication		
C1-CI-02-83		The specific implementation is not applicable to SQL Server.
C1-CI-02-86		The specific implementation is not applicable to SQL Server.
C1-CI-04-86		The specific implementation is not applicable to SQL Server.
I-0001	Delayed enforcement of authorization change	The TFM describes how privileges can be immediately revoked.
I-0096	Blanking passwords	SQL Server accepts Windows NT authentication. No password data is entered.
I-0234	One-time authentication mechanisms can be acceptable	This interpretation represents no requirement.
I-0240	Passwords may be used for card input	The specific implementation is not applicable to SQL Server.
I-0288	Actions allowed before I&A	SQL Server does not allow TCB mediated actions to occur (except the act of logging on itself) before logging on.
I-0314	Password changes do not require authentication	This interpretation represents no requirement.
Object Reuse		
I-0041	Object reuse applies to all system resources	All resources available at the TCB interface have been subject to residual information analysis.
Security Features User's Guide		
I-0244	Flexibility in packaging SFUG	The SFUG is appropriately packaged.
Security Testing		

	I-0170	Functional tests required for object reuse	Object reuse will be included as a security mechanism covered by the vendor’s test plans.
System Architecture			
	I-0213	Administrator interface is part of TCB	All evaluated administrator tools are considered part of the TCB and are described in the TFM.
System Integrity			
	I-0144	Availability of diagnostics	Machine specific diagnostics tools are part of the Windows NT TCB. Diagnostics specifically for the OS are available from Microsoft per the Windows NT TFM instructions.
Test Documentation			
	I-0281	Testing System Architecture functions	TCB protection will be included in the vendor’s test plans.
Trusted Facility Manual			
	I-0046	Detailed audit record structure	The stored audit structure is described in the TFM
	I-0069	Flexibility in packaging TFM	The TFM is appropriately packaged.

Appendix A List of Acronyms

ACL	Access Control List
ANSI	American National Standards Institute
API	Application Programming Interface
BCP	Bulk Copy Program
BDC	Backup Domain Controller
DAC	Discretionary Access Control
DBCC	Database Consistency Checker
DBID	Database Identifier
DBO	Database Owner
DBMS	Database Management System
DLL	Dynamically Linked Library
DMO	Distributed Management Objects
EC	Execution Context
GAM	Global Allocation Map
IAM	Index Allocation Map
ID	Identifier
I/O	Input/Output
IPAR	Initial Product Assessment Report
ISO	International Standards Organization
ITCSEC	Interpreted TCSEC
LC	Log Cache
LLF	Logical Log File
LLF_ID	LLF Identification
LPC	Local Procedure Call
LSN	Log Sequence Number
MB	Megabytes
MinLSN	Minimum LSN
MSDN	Microsoft Developer Network
MTF	Microsoft Tape Format
NTFS	NT File System
ODS	Open Data Services
OLE	Object Linking and Embedding
OS	Operating System
PAL	Publication Access List
PDC	Primary Domain Controller
PFS	Page Free Space
SFUG	Security Features Users Guide
SGAM	Small Global Allocation Map
SID	Security Identifier
SQL	Structured Query Language
TCB	Trusted Computing Base
TCSEC	Trusted Computer System Evaluation Criteria
TDI	Trusted Database Interpretation
TDS	Tabular Data Stream
TEF	TTAP Evaluation Facility
TFM	Trusted Facility Manual
TRB	Technical Review Board
TSQL	Transact SQL
TTAP	Trust Technology Assessment Program
UMS	User Mode Scheduling

Appendix B System Tables

System Table Descriptions

The SQL Server system tables are kept in four system databases:

- `master` database - The `master` database contains information on login accounts, system configuration settings, the existence of other databases, and initialization information for user databases and for SQL Server itself.

- `tempdb` database - The `tempdb` database provides server global temporary storage for temporary tables, stored procedures, and system worktables. The `tempdb` is accessible to all users as a scratch area. It is treated as any other database except that it does not persist from one SQL Server session to another.

- `model` database - The `model` database is used as an initialization template for all databases created by SQL Server.

- `msdb` database - The `msdb` database supports SQL Server Agent in scheduling alerts and jobs, and in recording operators.

System tables fall into two categories: those that contain global information and those that contain information specific to a single database within the server. The following are general descriptions of the contents of those tables. The descriptions are broken down into seven tables:

- Table B-1 Master Database Tables describes server-level system information tables that exist only in the `master` database;
- Table B-2 SQL Server Agent Tables describes server agent tables that exist only in the `msdb` Database;
- Table B-3 Database Backup and Restore Tables describes database backup and restore tables that exist only in the `msdb` database;
- Table B-4 Replication Tables in Master Database describes replication related tables that exist only in the `master` database;
- Table B-5 Replication Tables in Distribution Database describes replication tables that reside in the distribution database;
- Table B-6 Database System Tables describes database system tables that reside within every database; and
- Table B-7 Replication Tables in User's Database describes replication tables that reside within the user's database.

The SQL Server TFM specifies that access to system tables is to be restricted to the System Administrator.

Global Tables

Master Database Tables store server-level system information.

TABLE	CONTENTS
sysaltfiles	Describes files in a database.
syscacheobjects	Describes how cache is being used and provides look-up keys.
syscharsets	Describes character sets and sort orders.
sysconfigures	Describes each configuration option set by a user.
syscurconfigs	Describes each current configuration option.
sysdatabases	Describes databases on SQL Server system.

TABLE	CONTENTS
sysdevices	Describes disk backup file, tape backup file, and database file.
syslanguages	Describes each language on SQL Server.
syslockinfo	Describes all granted, converting, and waiting lock requests. This table is a tabular view of the internal data structures of the lock manager.
syslogins	Describes each login account, including server user ID and security identifier (SID). Tracks whether an account is an NT login or a SQL Server login. Also tracks whether an account is a member of server roles (including the `sysadmin` role).
sysmessages	Describes possible SQL Server messages.
sysoledbusers	Describes OLE linked servers.
sysperfinfo	Maintains internal performance counters.
sysprocesses	Describes processes currently running. Includes user who issued command and database being accessed. Identifies subthreads operating on behalf of single process.
sysremotelogins	Describes each remote user who is allowed to call remote stored procedures. Includes Windows NT user security ID.
sysservers	Describes each server that SQL Server can access as an OLE DB data source.

Table B-1 Master Database Tables

Server Agent Tables in the msdb Database contain data used by SQL server agents.

TABLE	CONTENTS
sysalerts	Describes alerts, which can take the form of various types of messages or generated tasks.
syscategories	Describes the categories used by SQL Server Enterprise Manager to organize jobs, alerts, and operators.
sysdownloadlist	Holds the queue of download instructions for all target servers.
sysjobhistory	Describes the execution of scheduled jobs by SQL Server Agent.
sysjobs	Describes each scheduled job to be executed by SQL Server Agent.
sysjobschedules	Contains schedule information for jobs to be executed by SQL Server Agent.
sysjobservers	Stores the association or relationship of a particular job with one or more target servers.
sysjobsteps	Describes each step in a job to be executed by SQL Server Agent.
sysnotifications	Describes each notification to be processed in response to an alert.
sysoperators	Describes each system operator to be notified in response to an alert.
systargetservergroupmembers	Records which target servers are currently enlisted in a multi-server group.
systargetservergroups	Records which target server groups are currently enlisted in a multi-server environment.
systargetservers	Records which target servers are currently enlisted in a multi-server operation domain.
systaskids	Contains a mapping of tasks created in earlier versions of SQL Server to SQL Server Enterprise Manager jobs in the current version.

Table B-2 SQL Server Agent Tables

Database Backup and Restore Tables are stored in the msdb Database. These tables store information used by database backup and restore operations.

TABLE	CONTENTS
backupfile	Describes each data or log file that is backed up.
backupmediafamily	Describes each media family for backup.

TABLE	CONTENTS
backupmediaset	Describes each backup media set.
backupset	Describes each backup set.
restorefile	Describes each restored file, including files restored indirectly by filegroup name.
restorefilegroup	Describes each restored filegroup.
restorehistory	Describes each restore operation.

Table B-3 Database Backup and Restore Tables

These tables are used by replication and stored in the master database.

TABLE	CONTENTS
sysarticles	Describes each article defined in the local database.
sysdatabases	Describes databases on SQL Server system.
sysobjects	Describes each object (constraint, default, log, rule, stored procedure, and so on) created within a database.
syspublications	Describes each publication defined in the database.
sysreplicationalerts	Describes information about the conditions causing a replication alert to fire.
sysservers	Describes each server that SQL Server can access as an OLE DB data source.
syssubscriptions	Describes each subscription in the database.

Table B-4 Replication Tables in Master Database

These tables are used by replication and stored in the distribution database.

TABLE	CONTENTS
MSagent_parameters	Contains parameters associated with an agent profile. The parameter names are the same as those supported by the agent.
MSagent_profiles	Describes each defined replication agent profile.
Msarticles	Describes each article being replicated by a Publisher.
Msdistpublishers	Describes each remote Publisher supported by the local Distributor.
Msdistributiondbs	Describes each distribution database defined on the local Distributor.
Msdistribution_agents	Describes each Distribution Agent running at the local Distributor.
Msdistribution_history	Contains history rows for the Distribution Agents associated with the local Distributor.
Msdistributor	Contains the Distributor properties.
Mslogreader_agents	Describes each Log Reader Agent running at the local Distributor.
Mslogreader_history	Contains history rows for the Log Reader Agents associated with the local Distributor.
MSmerge_agents	Describes each Merge Agent running at the Subscriber.
MSmerge_history	Contains history rows for previous updates to Subscriber.
MSmerge_subscriptions	Describes each subscription serviced by the Merge Agent at the Subscriber.
Mspublication_access	Describes each SQL Server login that has access to the specific publication or Publisher.
Mspublications	Describes each publication that is replicated by a Publisher.
Mspublisher_databases	Describes each Publisher/Publisher database pair serviced by the local Distributor.
Msreplication_objects	Describes each object that is associated with replication in the Subscriber database.
Msreplication_subscriptions	Contains one row of replication information for each Distribution Agent servicing the local Subscriber database.
MSrepl_commands	Contains rows of replicated commands.

TABLE	CONTENTS
MSrepl_errors	Contains rows with extended replication agent failure information.
Msrepl_originators	Describes each updateable Subscriber from which the transaction originated.
MSrepl_transactions	Describes each replicated transaction.
MSrepl_version	Describes the current version of replication installed.
Mssnapshot_agents	Describes each Snapshot Agent associated with the local Distributor.
Mssnapshot_history	Contains history rows for the Snapshot Agents associated with the local Distributor.
Mssubscriber_info	Describes each Publisher/Subscriber pair that is being pushed subscriptions from the local Distributor.
Mssubscriber_schedule	Contains default merge and transactional synchronization schedules for each Publisher/Subscriber pair.
Mssubscriptions	Describes each subscription serviced by the local Distributor.
Mssubscription_properties	Describes parameter information for pull Distribution Agents.

Table B-5 Replication Tables in Distribution Database

Database Local Tables

Database System Tables are stored in every database.

TABLE	CONTENTS
sysallocations	Describes each allocation unit.
syscolumns	Describes every column in every table and view, and a row for each parameter in a stored procedure.
syscomments	Contains entries for each view, rule, default, trigger, CHECK constraint, DEFAULT constraint, and stored procedure. The text column contains the original SQL definition statements, which are limited to a maximum size of 4 MB.
sysconstraints	Contains mappings of constraints to the objects that own the constraints.
sysdepends	Contains dependency information between objects (views, procedures, and triggers), and the objects (tables, views, and procedures) contained in their definition.
sysfilegroups	Describes each filegroup in a database. There is at least one entry in this table that is for the primary filegroup.
sysfiles	Describes each file in a database. This system table is a virtual table; it cannot be updated or modified directly.
sysforeignkeys	Contains information regarding the FOREIGN KEY constraints that are in table definitions.
sysfulltextcatalogs	Lists the set of full-text catalogs.
sysindexes	Describes each index and table in the database.
sysindexkeys	Contains information for the keys or columns in an index.
sysmembers	Describes each member of a database role.
sysobjects	Describes each object (constraint, default, log, rule, stored procedure, and so on) created within a database.
syspermissions	Contains information about permissions that have been granted and denied to users, groups, and roles in the database.
sysprotects	Contains information about permissions that have been applied to security accounts with the GRANT and DENY statements.
sysreferences	Contains mappings of FOREIGN KEY constraint definitions to the referenced columns.
systypes	Describes each system-supplied and each user-defined data type.

TABLE	CONTENTS
	These are the system-supplied data types and their ID numbers.
sysusers	Describes each Windows NT user, Windows NT group, SQL Server user, or SQL Server role in the database.

Table B-6 Database System Tables

These tables are used by replication and stored in the user's database.

TABLE	CONTENTS
MSmerge_contents	Describes each row modified in the current database since it was published. This table is used by the merge process to determine the rows that have changed.
MSmerge_delete_conflicts	Contains information for rows that were deleted because either they conflicted with an update and lost the conflict or the delete was undone to achieve data convergence. This table is stored in the database used for conflict logging, usually the publishing database, but can be the subscribing database if there is decentralized conflict logging.
MSmerge_genhistory	Describes each generation that a Subscriber knows about (within the retention period). It is used to avoid sending common generations during exchanges and to resynchronize Subscribers that are restored from backups.
MSmerge_replinfo	Describes each subscription. This table tracks internal information about the sent and received generation.
MSmerge_tombstone	Contains information on deleted rows and allows deletes to be propagated to other Subscribers.
sysarticleupdates	Describes each article that supports immediate-updating subscriptions.
sysmergearticles	Describes each merge article defined in the local database.
sysmergepublications	Describes each merge publication defined in the database.
sysmergeschemachange	Contains information about the published articles generated by the Snapshot Agent.
sysmergesubscriptions	Describes each known Subscriber and is a local table at the Publisher.
sysmergesubsetfilters	Contains join filter information for partitioned articles.

Table B-7 Replication Tables in User's Database

Appendix C Evaluated Components

C.1 Operating System Components

Windows NT Server and Workstation 4.0 Service Pack 6a and C2 Update are the operating systems in the evaluated configurations. These product configurations encompass all TCB hardware and software that is included in the evaluated configurations as described in the **Windows NT 4.0 Service Pack 6a Final Evaluation Report**.

C.2 SQL Server Components

The evaluated software configuration includes SQL Server 2000 version 8.0 as configured by the TFM.